UQAC
Université du Québec
à Chicoutimi

AUTOMATED FRAMEWORK FOR COMPREHENSIVE DIFFICULTY EVALUATION OF ENEMIES USING BEHAVIOUR TREES

PAR DAVID PONTON

MASTER THESIS PRESENTED TO L'UNIVERSITÉ DU QUÉBEC À CHICOUTIMI IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN THE SUBJECT OF INFORMATIQUE - 3017

QUÉBEC, CANADA

ABSTRACT

Correctly tuning difficulty in video games is a task that may appear arbitrary on the surface, but is actually one of the major foundations of game design, as it directly affects the level of player engagement. Unfortunately, this important step is presently only possible after the game is in a playable state, and this remains true in the era of AI and machine learning (ML) as even these advanced tools rely on human-provided data to function. This master's thesis attempts to circumvent this requirement by identifying difficulty metrics which are game-centric rather than player-centric. Video game difficulty consists of various types, including comprehensive difficulty, which stems from the various rules and parameters that make up the game. This includes the behaviour of in-game enemies, which this master's thesis breaks down into a formal metric for analysis. In order to achieve this, I propose an innovative framework which takes advantage of the common game design practice of modelling non-player characters (NPC) into graphs known as Finite State Machines (FSM) or Behaviour Trees (BT). Our approach leverages graph properties as well as the standardized structure of these two models in order to produce a metric combining the amount of information in the graph, defined by the number of nodes, and the amount of coupling between it, measured by the graph's cyclomatic complexity. Moreover, our framework presents a method for homogenizing the morphology of structures before their analysis, through a set of algorithms which convert FSMs and BTs into one another. By converting the original FSMs into BTs and the original BTs into FSMs and then back into BTs, we can unify both models into what we refer to as canonical BTs, a model with a predictable topology. This ensures compatibility with the rest of our framework and allows for direct comparison of enemies regardless of which model they are represented in. To validate our method, I performed a case study consisting of various enemies from well known games such as *Super Mario Bros.* and *Mega Man* in which I processed each enemy's behavioural graph with the aforementioned algorithm and then compared them against each other. The master's thesis concludes by discussing the limitations and the potential of this method as well as pointing out future work which could mitigate the former and capitalize on the latter.

RÉSUMÉ

Ajuster correctement la difficulté dans les jeux vidéo est une tâche qui peut sembler arbitraire à première vue, mais qui constitue en réalité l'un des fondements majeurs du game design, car elle influence directement le niveau d'engagement des joueurs. Malheureusement, cette étape essentielle n'est actuellement possible qu'une fois le jeu dans un état jouable, et cela demeure vrai après l'avènement de l'IA et de l'apprentissage machine (ML), puisque même ces outils avancés reposent sur l'usage de données fournies par des humains. Cette thèse tente de contourner cette contrainte en identifiant des métriques de difficulté centrées sur le jeu plutôt que sur le joueur. La difficulté dans les jeux vidéo se divise en plusieurs types, dont la difficulté de compréhension, qui découle des différentes règles et paramètres présentes dans le jeu. Cela inclut notamment le comportement des ennemis, que cette thèse décompose en une métrique formelle pour l'analyser. Pour ce faire, je propose un cadre novateur qui s'appuie sur une pratique courante dans le game design: la modélisation des personnages non-joueurs (NPC) sous forme de graphes, soit les Machines à États Finis (FSM) et les Arbres de Comportement (BT). Notre approche exploite les propriétés des graphes ainsi que la structure normalisée de ces deux modèles afin de produire une métrique qui combine la quantité d'information dans le graphe, définie par son nombre de noeuds, et le degré de couplage parmi celle-ci, mesuré par la complexité cyclomatique du graphe. De plus, notre cadre inclut un méthode pour homogénéiser la morphologie des structures avant leur analyse, grâce à un ensemble d'algorithmes permettant la conversion réciproque des FSMs et des BTs. En convertissant les FSMs en BTs ainsi que les BTs en FSMs puis à nouveau en BTs, nous pouvons unifier les deux modèles en ce que nous appelons des BTs canoniques, un modèle à topologie prévisible idéal pour notre cadre car il permet la comparaison directe des ennemis peu importe utilisé pour les représenter. Afin de valider notre méthode, j'ai réalisé une étude de cas sur différents ennemis issus de jeux connus tels que *Super Mario Bros.* et *Mega Man*, dans laquelle j'ai traité le graph comportemental de chaque ennemi à l'aide des algorithmes ci-dessus, puis les ai comparés entre eux. La thèse se conclut par une discussion sur les limitations et le potentiel de cette approche, ainsi que sur les perspectives de recherche future dans ces deux optiques.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

FSM Finite State Machine

BT Behaviour Tree

ML Machine Learning

NPC Non-Player Character

ACKNOWLEDGEMENTS

I'd like to thank my directors Hugo Tremblay, Bruno Bouchard and Yannick Francillette for guiding me in the right direction whenever I wasn't sure on how to proceed. I'd also like to thank my parents for supporting and encouraging me throughout this journey, and my friends for helping me stay focused when I got sidetracked.

DISCLAIMER ON USAGE OF AI

Some of the content of this thesis was partially produced by AI. ChatGPT was used to brainstorm and provide inspiration for topics to cover in the introduction, to generate a list of the key elements from each article reviewed in Chapter 3 to help me structure their summaries better, and also to approximate the original behaviour trees of the various enemies analyzed in Chapter 5 in an attempt to keep the results clear of my own design choices. Elicit was also used to find some of the references used in Chapter 3.

CHAPTER I

INTRODUCTION

Over the last few decades, video games have gone from being a novel but rudimentary form of home entertainment to being a widespread and influential medium deeply entrenched within modern culture. Nowadays, the video game industry rivals or even surpasses that of more traditional media in profitability and expanse. According to a report by Precedence Research, the global video game market was valued at approximately USD 274.63 billion in 2024, and is predicted to reach over USD 700 billion within the next 10 years[1]. A Reuters article cites another report by Newzoo which claims that the global gamer community sat at an impressive 3.42 billion people in 2024[2]. This is not that surprising, as video games have received a massive influx of popularity in recent years, in one part with the advent of ever more powerful phone models, but also likely due to people having to stay at home more during the COVID years. What used to be a scary new technology for concerned parents is now a fun hobby enjoyed by the whole family, as the medium is now much more mainstream and widely accepted by the general population. Where video games were once much harder to get into and their user base consisted mostly of passionate but reserved gamers, they are now an extremely social platform with a powerful influence on modern culture. Even other industries find themselves unable to ignore it as advertisement space is a common method for free-to-play games to finance themselves.

Part of this switch in the culture surrounding video games is thanks to several studies published over the years, which sought to prove that video games can have multiple benefits outside of mere entertainment. Over time, video games have ceased being cast in a negative light and the health-conscious demographic stopped seeing them as majorly destructive for today's youth. Of course, moderation remains important, but for instance, there is now ample

evidence that both children and elderly individuals who play action or strategy video games develop and maintain better cognitive performance in various ways, including memory, spatial ability, task switching, and visual perception[3, 4, 5, 6, 7, 8]. Additionally, reports show that playing multiplayer strategic video games help develop communication and teamwork skills, and that social video games in general help foster a sense of community and belonging as well as provide a safe space for social interaction for individuals with social or autism disorders[9, 10, 11].

Today, nearly everybody enjoys some form of digital gaming, whether it's a casual time killer on a mobile phone or a competitive hobby on the computer. From cute story games for children to cut-throat combat games such as *Dark Souls*, there is something for everybody. However, what exactly makes video games fun? A lot of it comes down to engagement, and engagement in a video game is directly proportional to how well it flows. This, in turn, depends on whether the game is an appropriate challenge for the player. Indeed, Csikszentmihalyi's Flow theory[12] suggests that players become frustrated when a game is too difficult but also become bored when the game is too easy. For this reason, proper balancing of a video game's difficulty is instrumental in providing a desirable experience to the target audience. Frampton[13] suggests that difficulty in games can be broken down into three types: motor (also referred to as executive or physical), which relates to the player's motor skills, strategic, which tests the player's decision-making ability, and comprehensive, which represents more of an overhead difficulty in understanding the game and how to play it. An article by Denisova et al.[14] also offers emotional difficulty and cognitive difficulty as two other types, stating the former as pertaining to the player's ability to make choices in a story based games, and the latter as relating to the player's cognitive abilities such as memory, observation and problem solving.

Figure 1.1 : Super Meat Boy, a precise and fast-paced platformer

Motor difficulty comes largely from the degree of precision and speed required in terms of controller input. Fast-paced games with complex button combos, difficult manoeuvres, and tight reaction timings are the type of games with high motor difficulty. Examples of such games include rhythm games like *Guitar Hero* or *Osu* and difficult platformers such as *Super Meat Boy* or *Celeste*. Strategic difficulty, as the name implies, primarily relates to strategy games. On the other hand, turn-based games where every minute decision matters and where operational optimization can be pushed to great lengths to make the difference between success and defeat against all odds are the games where strategic difficulty is the most prominent. These types of games include 4X[1] strategy games like the *Civilization* series or highly decision-based rogue-likes such as *FTL: Faster Than Light* or *Slay the Spire*.

[1] eXplore, eXpand, eXploit, eXterminate

Figure 1.2 : Civilization VII, a very complex 4X strategy game

These two types of difficulty tend to be the most prominent types in competitive games, and the amount of each present in a game varies wildly across genres. Games which require a high degree of motor skill tend to advertise it as their main focus, such as First Person Shooters (FPS) or fighting games, which greatly reward accuracy and precise timing, respectively. Likewise, players who excel at these generally want to test those skills only and often find deep strategy to be secondary or even bothersome. A major downside of physically demanding games is that they become unplayable for players with physical disabilities, something which does not happen with strategic games. Indeed, games which focus on strategic difficulty are often turn based or at least slow paced, making them much more accessible, as with enough time and practice, anyone could grasp even the most complex strategy game, but someone without full control of their hands can simply never reach a high level in fast paced games. Another argument against physically-intensive games is that even an extremely wise and

Figure 1.3 : Slay the Spire, a strategy card game which rewards strong decision-making skills

tactical player could make the correct decision but still end up making a mistake if they input incorrectly, something which is common at the extremely high levels of Real Time Strategy (RTS) games or tactical shooters such as *Starcraft* or *Counterstrike*. This often leaves the player in a state of frustration as they feel "robbed" them of an optimal decision.

Emotional and cognitive difficulty are typically more befitting of casual games. Visual novels with multiple endings based on player choices or grim, cut-throat RPGs where characters are expected to perish and sacrifices must sometimes be made, a common trope in horror games such as *Fear and Hunger*, are the perfect example of games where emotional difficulty shines through. Cognitive difficulty, on the other hand, is most of the time found in arcade-style puzzle games often found on mobile platforms, *Candy Crush* being a popular example. They typically don't feature the same long term vision and planning that strategy games involve, but still require correct decision making in the short term, meaning they also

reward acute observation in order to spot what the best decision may be in a given situation. Of course, strategy games themselves frequently involve cognitive difficulty too, for instance in the player's ability to remember things they've seen, such as what the enemy player is doing after having spied on them. However, some strategy games abstract from such difficulty by implementing tools to summarize information from which to let the player more easily make their decisions. For example, *Slay the Spire* is a strategic card game which allows the player to view their discard and draw piles at any point rather than forcing them to remember what cards they've played in earlier turns.

But what about comprehensive difficulty? What kind of game genre wants to be difficult to understand? Is it not generally good design for a game to be easy to grasp? While that may be the case, one can only make a concept so palatable if it is very complex, and complexity is a major factor in all types of difficulty. A strategy game that is too simple would likely be boring, as would an action game with too few possible actions, or a platformer which only ever features the same type of obstacles. In all of these cases, there are ways to make the game artificially harder. For instance, the designer of the strategy and action games could give the enemies resource or combat advantages, and the platforming game's designer could make the the levels very long and force you to start over completely upon failing. Is that particularly fun, however? Online articles generally suggest otherwise, or at least that it generates frustration for the player[15, 16, 17]. This is why comprehensive difficulty constitutes an important backbone of the difficulty spectrum, as while it isn't a type of difficulty that defines any particular genre, it is simply a by-product of the complexity required by the other difficulty types.

Unfortunately, being omnipresent within a game makes comprehensive difficulty challenging to gauge and calibrate, as every single system and concept within a game contributes to it, often in a tangled web of various relations. One particularly important and obvious

element when it comes to difficulty of any kind in most games is the enemies. Enemies are the main way that many games provide adversity against the player and are therefore often the main source of challenge. This means that defeating them yields engagement, satisfaction, and fun for the player. As mentioned previously, a certain balance of challenge must be maintained in order for the game to remain fun, and in many cases, this means that enemies must have at least some complexity to them. After all, few players would enjoy simply hitting target dummies that don't fight back or even attempt to evade hits. It is therefore essential for the enemy AI to exhibit intelligent and immersive behaviours in order to keep the player in a proper state of flow.

Typically, enemy difficulty scales upward as the player progresses through a game. Simplistic design may achieve this by making the later enemies numerically stronger, however a well designed game should instead opt to make the enemies have increasingly complex behaviours. They could have new abilities, combine multiple existing abilities, employ better tactics or simply behave smarter. When enemies learn to behave in ways the player hasn't seen before, it creates excitement as it presents them with a new "puzzle" to figure out[18]. The comprehensive difficulty of an enemy can be caused by a myriad of factors, such as having a wide array of possible behaviours, having multiple forms, weapons, etc., or having specific conditions which trigger special events, for instance retreating when near death.

Correct calibration of a video game's difficulty level requires a great deal of finesse in the form of the slow and intensive process that is repetitive user testing. Tuning the enemies' strength is no exception to this, and most traditional methods must wait until near the end of the project before they can begin, often leading to lengthy extensions of production time. The usage of Machine Learning (ML) has been proposed to accelerate this process[19, 20]. When applied to other types of difficulty, this model is mostly used to dynamically adjust difficulty as the player progresses through the game, however this technique does not apply particularly

well to comprehensive difficulty. At best, the ML code could be trained to rate the game's comprehensive difficulty but could not automatically adjust it. In addition, even such a method would still require real players to play-test the game in order to generate data. Therefore, this does not solve the issue that the testing must wait until the video game is playable and mostly finished, as the game still needs to be played for data to exist. ML merely lets designers draw conclusions from this data quicker and with less effort. Moreover, ML solutions typically rely on purely numerical adjustments rather than design changes such as the removal, modification or addition of meaningful content. For instance, ML in its current form would not suggest the addition of a new ability to the player in order to make certain encounters easier. It would instead simply suggest lowering the enemies' damage or health and call it a day. This style of ML is therefore better fit for long term game balance in Games as a Service (GaaS)[21], which are games that continue receiving new content and updates long after their release, such as *World of Warcraft* or *Dota 2*. For this reason, there is still an unfilled need for non-ML methods of difficulty assessment that can be used not only earlier in the cycle but also in an "agile" fashion at a moment's notice, such as when adding a new enemy or a new system, in order to immediately understand its impact on the game's difficulty. An additional benefit of such a tool would be that it could be implemented as a plugin or add-on in existing commercial game engines.

When it comes to measuring the various types of difficulty, a large selection of factors contribute to each. One important aspect when selecting which factors to include in the model is to consider whether each metric is player-centric or game-centric. Dynamic Difficulty Adjustment (DDA) is, by its very nature, driven by player-centric data, meaning that the dataset is subjective to each player. Examples of such metrics include how many times the player has died, their score, the time they take to complete a level, etc., which vary significantly between each player depending on their skill level. While this may be sufficient for real-time

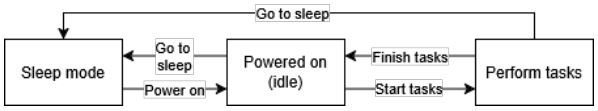

Figure 1.4 : Example of a simple Finite State Machine

difficulty tuning, this type of data is ill-fitted for a more objective and preemptive approach, where game-centric metrics are required in order to accurately determine the level of difficulty, not to mention that player data is, once again, unavailable early in the development cycle. Game-centric metrics include, for instance, the number of enemies in a level, the size of the level, the player character's abilities, etc. For example, the first level of *Super Mario Bros.* has a specific degree of difficulty which can be calculated and, when compared to other results from the rest of the levels or even other games, would indicate that it is objectively among the easiest. This does not mean that it is easy for everyone, as younger or inexperienced players may very well still struggle to complete even the easiest levels; all that matters is that the line for "easy" is drawn somewhere and that this line is consistent.

Unfortunately, in the current literature, not much work has been done towards identifying an automatic mean of evaluating the difficulty of enemies[22, 23] nor towards establishing objective definitions of said difficulty. Earlier work by Francillette et al.[24] brought forth a metric for assessing enemies' comprehensive difficulty through their modelisation as a Finite State Machine (FSM), a type of graph used to describe the various states of a system and the possible transitions between them. For example, a robot which can go to sleep when it's powered on, power on when it's asleep, and also perform tasks while it's powered on would have the sleep mode state, a powered on but idle state, and then the perform tasks state, with bidirectional transitions between each one of those except for one going from sleep to performing tasks. See Figure 1.4 for a visual representation of this.

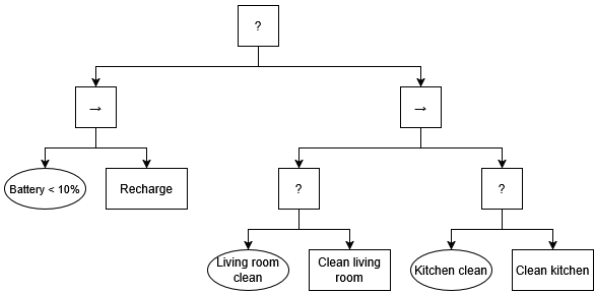

Figure 1.5 : Example of a simple Behaviour Tree

FSMs are one of two primary ways enemies' behaviour is commonly represented in the video game industry, the other being Behaviour Trees (BT). BTs are trees where each node represents either instructions for control flow or a behaviour to execute, with the former being parent nodes and the latter being the leaves. To make an example without going too deep in the specifics just yet, if a robot was programmed to clean up various rooms in a house but also prioritize recharging its battery if it's running low, a BT representing this would consist of two main branches splitting from an "OR" control node at the root, with the branch on the left checking if the battery is low and then executing a recharge task if it is, and the branch on the right iterating through every room in the house with an "AND" node and checking if they need to be cleaned and cleaning them if that is the case. A visual representation of this is shown in Figure 1.5.

This master's thesis covers my journey in learning about the various important concepts of graphs, FSMs and BTs and the literature which covers them throughout my master's degree. The goal is to contribute to the understanding of how to automatically evaluate video game enemies' comprehensive difficulty. My master's thesis, whose contributions were featured in an IEEE conference as well as an Elsevier Computing journal article, does so by exploring the different methods used describe video game enemies and lays down the groundwork on how to convert an enemy's representation back and forth between a FSM model and BT model in a

canonical fashion which produces consistent results. In it, I propose two algorithms, one for each direction of the conversion, based on existing literature on which I expand by offering potential solutions to the intricacies that arise when various node types are used. The use of a standardized structure model enables the reliable use of key metrics regardless of which model type is used. The two metrics are the volume of information present in the structure and its complexity, measured by the amount of coupling between the structure's different nodes. This, in turn, lets us establish a correlation between these metrics when applied to FSMs and when applied to BTs, allowing the objective comparison of enemy difficulty no matter which model is used, which constitutes an early step towards automatic evaluation and a reduction in the need for play-testing.

The following sections are found within this master's thesis: the next chapter contains a summary of the prerequisite notions for understanding the subject. Chapter three reviews the current scientific literature on the topic as well as in other adjacent and relevant spheres. Then, chapter four follows with a deeper explanation of my proposed solution to the problem and how it was put together. Lastly, chapter five demonstrates the solution with a case study and then provides a list of the results found throughout the experimentation process along with a discussion on the findings.

CHAPTER II

THEORETICAL BACKGROUND

This chapter's purpose is to cover all the necessary bases for the reader to understand the rest of the thesis's contents. It also includes some adjacent notions that are not directly related to the work detailed in the thesis but that I've deemed relevant for potential future work. The topics covered include: a formal definition of video games, the concept of flow, an explanation of difficulty types in games, and finally, a deep dive into the various properties, definitions and components of the graphs used to model character behaviour.

2.1 VIDEO GAMES

According to Tavinor[25, 26], a formal definition of video games is as follows:

> "X is a videogame iff it is an artefact in a digital visual medium, is intended primarily as an object of entertainment, and is intended to provide such entertainment through the employment of one or both of the following modes of engagement: rule-bound gameplay or interactive fiction."

In other words, video games are pieces of software intended as interactive entertainment. They are developed following similar processes as any other software. They start with design documents detailing various aspects of the finished product, such as the interface, the gameplay, the art style, examples of existing similar products, etc., though Game Design Documents (GDD) are typically less technical than the documents used in classic software development. Following the design phase, development generally also resembles that of normal software; Agile methods are common, though waterfall-style methodologies also exist. When using

Agile, the development team plans the entire project by splitting it into several user stories, typically each corresponding to one of the game's features or an activity or task the player can do within the game. The team then plans sprints every so often by selecting a number of stories to complete for that sprint, depending on the team's pace which will have been gauged beforehand. The development team, much like that of classic software, tends to consist of a variety of specialists, of course including developers who focus on the game engine, the user interface, characters, game mechanics, world interactivity, etc., but also non-programmers like audio designers, artists, writers, game designers, etc. who each also have specialties such as character artists versus environment artists, or designers who manage the game balance versus those who take care of level design. All of these roles require a lot of work; game design is no exception to this.

Game design, particularly when it comes to game balance, has an unusual particularity to it in that it is generally the job that both begins and does the finishing touches to a game project. Game designers are the ones who put together the GDD, but they are also the ones who must make sure to correctly tune the game once it's nearing completion. This is done through a typically very large number of playtesting sessions, which tends to significantly draw out the completion time of a game development project. Not only is it a long and iterative process in which data must be collected from play-testers in order to calibrate the game which must then be tested again, but this process can only start once the game is in its polishing phase, meaning it's playable and possesses just about all of its planned features. Playtesting can still be done beforehand, though in that case it is generally to gauge whether a particular design is liked by the players more so than to gauge the balance or difficulty of the game.

2.1.1 FLOW

As mentioned previously, entertainment is an important factor in what defines a video game, like one might expect; and as covered earlier, one element that plays a big part in ensuring said entertainment is the state of flow. In the proceedings of the 2014 Conference on Interactive Entertainment, Velikovsky[27] summarizes Csikszentmihalyi's theory of flow[28, 29, 30] as "essentially a theory of happiness, deep enjoyment, or, 'fun' for those immersed in a given task." In 1996, Csikszentmihalyi formulated nine key characteristics that indicate somebody is in a state of flow[31]. Among the more relevant ones are: there are clear goals every step of the way, there is immediate feedback to one's actions, there is a balance between challenges and skills, and there is no worry of failure. In other words, the key components of maintaining a state of flow are immersion, a sense of control and assurance but also a sense of pride in doing something that isn't trivial or that could be done by just anybody. The player has to remain in a loop where they feel like their time getting good at the game so far was worth it because they are now good enough to play this next difficult level. They must also be kept guessing in a way; if they can see themselves beating a level but aren't one hundred percent sure what it's going to look like, then the task is satisfying as they get to prove to themselves that they can beat it. However, if they can already fully predict what the level is going to be like because they've mastered the game beyond what the level demands, then there is no more novelty, only a chore.

2.1.2 DIFFICULTY

In order to gauge the difficulty of a level, we must understand what difficulty means. There are multiple different types of difficulty, with the most commonly identified ones being motor or executive difficulty, strategic difficulty, and comprehensive difficulty[13], though a few others such as emotional difficulty and cognitive difficulty have been named as well[14].

Motor difficulty tests the player's physical abilities. This includes speed, dexterity, coordination and reaction time. Games with precise or complex controls or games which require fast reaction times are the type of games where this difficulty is prominent. A game like *Dark Souls* features a high degree of motor difficulty because the player must react to enemy attacks within a very short window in order to dodge or parry them. An example of a game where the motor difficulty comes from complex controls and coordination is *World of Warcraft*, where players are expected to evade enemy attacks by controlling their character's position through movement keys, while also continuing to rapidly use various abilities either on the enemy or their allies (generally referred to as a "rotation", meaning they rotate through abilities which must go through a short cool-down period before they are able to be reused), and also managing a different set of abilities with longer cool-down periods that can only be used once or twice in most encounters and must therefore be carefully used at the correct time. Next, First-Person Shooters like Counterstrike and Call of Duty are what comes to mind when thinking of motor difficulty resulting from precise controls; players are first and foremost challenged on whether they have accurate aim. Lastly, though it is rarely the main draw of a game, there are many games whose communities enjoy pushing themselves with further challenges where speed is in fact the main difficulty; one might think of Super Mario World rom-hacks or "speedrun" levels in Super Mario Maker. Another example is performing the most difficult songs on the hardest difficulty in Guitar Hero.

Strategic difficulty, on the other hand, challenges players' decision-making abilities, be it in the way of maximizing odds in a game of randomness, optimizing resource usage in a survival situation, or strategically deploying troops in a conflict on multiple fronts in a war game. One of the most obvious examples of a game with high strategic difficulty is the genre known as 4X strategy, which is abbreviation of Explore, Expand, Exploit, Exterminate, a name that speaks volumes on all the layers of strategy interlaced in the genre. Players'

decision making skills are tested at all stages of the game, between usage of troops to explore their starting area, balancing rapid expansion and strong infrastructure in order to not be left behind nor expose themselves to a brutal surprise attack, and properly exploiting whatever resources their land happens to have, all in order to eliminate their enemies as efficiently as possible. Another genre which features a significant amount of strategic difficulty is turn-based rogue-likes. Games like *Slay the Spire* or *Monster Train* where the player must replay through an increasingly difficult game result in the player needing to make increasingly optimal choices if they are to survive the game as the leeway between life and death becomes tighter and tighter. Rogue-likes are built on the concept of replaying a relatively short journey with randomized tools at the player's disposal, whether these are weapons, cards, power-ups, etc. depending on the game. However, these two card games feature a mechanic where an optional difficulty level increases every time the player manages to complete a "run" of the game on the highest unlocked difficulty. Earn increase in difficulty is small, but after several iterations, it starts to add up. On the highest difficulty, the game is ruthless; players can no longer afford to make speculative decisions in hopes of finding the right card to make a certain strategy work later. Their deck needs to work right now and they have to make do with what the game gives them and slowly transition from a short-term strategy to a long-term one which can defeat the final boss over the course of the run, weighing opportunity costs between decisions over and over. For instance, one of the possible encounters in *Slay the Spire* is a rest site where the player can choose to heal some of their hitpoints back, or upgrade one of their cards to make it better. If the player is too greedy and refuses to heal at a critical moment, they can get a bad surprise from one of the several "normal" encounters that are now deadly on their own in comparison to the base difficulty. However, if they play too safe and only prioritize staying alive and healing every chance they get rather than solidifying their game, the game will simply outscale them, and even if they do make it to the final boss, they may just not be able to defeat it.

Next comes comprehensive difficulty, which is the type relevant to this thesis. Comprehensive difficulty, unlike the rest, is almost never the main focus of a game; in fact, it is typically something games want to avoid. The easier it is to understand a game, the easier new players can get into it. However, it can never be completely avoided and very often, other types of difficulty require some degree of comprehensive difficulty in order to exist, as comprehensive difficulty mostly stems from the game's complexity. This is most obvious with strategic difficulty; it's hardly possible for a game to have high strategic difficulty without at least a moderate degree of complexity. There are many potential metrics to measure a system's complexity; Polančič and Cegnar list several in an article on process models[32], such as Lines of Code, Halstead complexity measures, and cyclomatic complexity, the last of which is what the contents of this thesis are based on. In order to use cyclomatic complexity as a metric, the systems under analysis must be represented as graphs.

2.2 GRAPHS

According to Gross and Yellen[33], the formal definition of a graph is a mathematical structure which consists of a set of vertices, denoted V, and a set of edges, denoted E. Each edge in E usually has a set of two vertices associated with it, which it connects; it is however possible for en edge to connect a vertex to itself, in which case it only has one vertex associated with it. When a graph G is not the only graph in a given context and the mere notations V and E become ambiguous, the notations V_G and E_G are used to specify that the sets belong to the graph G. When modifying graphs, the term subgraph is often used to designate a graph whose vertices and edges are all in another graph. In other words, if you took a graph G and removed some of its edges and vertices and named that new graph H, H would be a subgraph of G; however if you then added an entirely new edge or vertex to H that is not in G, then it would no longer be a subgraph of G.

By default, edges in graphs are not given a direction, and therefore can flow in either sense of the edge. A graph in which every edge has a specific direction is called a directed graph, or digraph. The direction of an edge is indicated by an arrow pointing at the forward vertex.

A walk is an alternating sequence of connected vertices and edges from a starting vertex v_0 to an end vertex v_n, and a path is a walk where no vertex or edge appear twice. A graph is said to be connected if there exists a walk between every pair of vertices in it, and a directed graph is said to be strongly connected if this remains true in both directions for all pairs when respecting edge direction. Each disjointed maximal connected subgraph in a graph G is referred to as a component of G, with "maximal" in this case meaning that no other vertex in G connects to the subgraph; by definition, any connected graph has only one component. This leads us to the definition of a cycle, also known as a closed path, which is a path where the endpoint is the same as the starting vertex. It also enables the definition of distance $d(s,t)$ between two vertices s and t, which is the length of, or the number of edges in the shortest walk between them. That is, if that walk exists; if it doesn't, then the distance is equal to ∞. From here, we can continue to the definition of a vertex's eccentricity in a graph G, which is the distance between itself and whichever other vertex in G is furthest from it. Lastly, the diameter and the radius of a graph are the maximum and the minimum of the graph's vertex eccentricities, respectively; the centre of the graph is the subgraph containing all vertices with minimum eccentricity along with any edges between them.

Figure 2.1 is a strongly connected graph of radius 2 and diameter 3: vertices a and c have an eccentricity of 3, this is due to the distance of 3 between a and d as well as between c and a, both of which are a's and c's longest distance to another vertex. Note that in a directed graph, distance is not reciprocal: for instance, the distance between d and a is only 2, which is the longest distance from d to another vertex, tied with its distance to c. Alongside it, b also

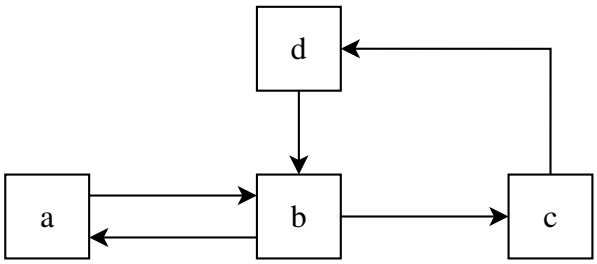

Figure 2.1 : An example graph

has an eccentricity of 2 through its distance to d, making the subgraph with $V = \{b, d\}$ the centre of the full graph.

2.2.1 TREES

Trees are a specific type of graph with the following characteristics: it must be connected and contain no cycle, meaning there is only one possible path between any two nodes. This also means that a tree with n vertices contains $n-1$ edges and that the deletion of any of them would split the graph in two disconnected sections. Likewise, it is impossible to add an edge between any two vertices of a tree without creating exactly one cycle.

A useful concept when making trees for decision making, artificial intelligence, and for describing behaviours in general is rooted trees. A rooted tree is a directed tree (meaning it is a directed graph that is also a tree) where one of the vertices is designated as the root. That vertex is the only vertex in the tree with an in-degree of 0, meaning it doesn't have any edges directed towards it; all of its adjacent edges flow outwards from it, whereas other vertices all have an in-degree of one. Rooted trees also also come with a few terms unique to them. The depth (or level) of a vertex is its distance from the root, and the height of the tree is its highest vertex depth, or in other words, the length of the longest path between the root and another vertex. A pair of vertices sharing an edge are called parent and child, with the latter being the

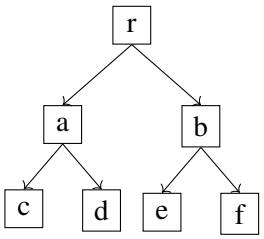

Figure 2.2 : An example tree

downstream vertex and the former being the one which precedes it. A leaf is a a vertex with no children, and an internal vertex is one with children. Figure 2.2 shows an example of a rooted tree with height 3; r is the root node, $\{c,d,e,f\}$ are the leaves and $\{a,b\}$ are their parents.

2.2.2 CYCLOMATIC COMPLEXITY

Cyclomatic complexity is the chosen metric in this project for estimating comprehensive difficulty. It represents the number of linearly independent circuits[34], also known as fundamental cycles[33], in a strongly connected graph. Another way to put it is the number of edges that can be removed from a graph G while keeping G connected; this number is known as the cycle rank of G, $\beta(G)$. The subgraph F resulting from the removal of the entire set of those edges from G is called a spanning tree of G, meaning it is a graph which shares the same set of vertices as G, but is a tree. The set of edges that were removed to achieve this is called the relative complement of F, denoted $G-F$, and the number of remaining edges in F is called the edge-cut rank of G and is equal to $|V_G|-c(G)$, where $c(G)$ is the number of components of G. This ties in with the definition of a fundamental cycle, which refers to the cycle created from adding any one edge e from the relative complement $G-F$ back into G. That cycle, by the definition of a tree seen earlier, is guaranteed to exist and is the only cycle in the subgraph $F+e$; it will also be different for each edge in $G-F$, and since there are $\beta(G)$ edges in $G-F$, there are therefore $\beta(G)$ fundamental cycles in G. In summary,

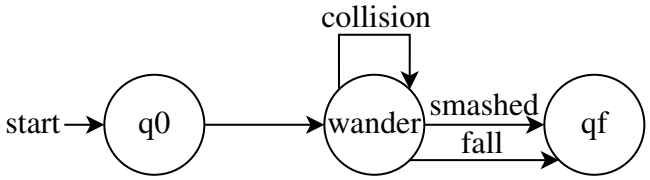

Figure 2.3 : FSM representing the Super Mario Bros. starter enemy, the Goomba.

$\beta(G) = |E_{G-F}| = |E_G| - (|V_G| - c(G))$, which, for a strongly connected graph is equivalent to $\beta(G) = |E_G| - |V_G| + 1$.

2.3 FINITE STATE MACHINES

Finite State Machines (FSM) are a basic mathematical model for computation, AI and robotics. They consist of a set of vertices called states and edges called transitions, as well as a set of events that cause each transition. Each state, as the name indicates, represents a state the machine can be in; for instance, a character may be in a walking or running state, or an attacking state. Each state can be accessed from a select number of other states through transitions which represent a condition which must be met for the agent to change to that state; for instance a character who is running may be forced to start walking if their stamina is low. When applied to video game enemies, they also typically feature an initial state q_0 which represents the enemy entering play or being loaded into the game, and a final state q_f which represents the enemy being destroyed or removed from the game. 2.3 shows a simple example of a FSM for the Goomba from *Super Mario Bros*. Like all FSMs, it begins by loading the entity, which in *Super Mario Bros.* is when the enemy appears on the screen. Then, its entire behaviour consists of wandering around, switching directions when colliding with an obstacle. Its destruction node is eventually reached either when it falls off the stage or when the player stomps on it.

FSMs are very popular as they are a fairly adaptable structure which can be used in many fields of computer science. They are also intuitive and easy both to read and implement. They do however present some significant disadvantages when used to model bigger systems. They do not scale particularly well as they quickly become messy, as in order to maintain connectivity (and therefore, the FSM's ability to properly react to stimuli), the number of transitions must grow quadratically with the number of states, which reduces modularity. In other words, larger FSMs suffer from a trade-off between reactivity and modularity.

2.4 BEHAVIOUR TREES

Behaviour trees (BT) are directed rooted trees that describe an array of tasks to be performed by an autonomous agent as well as the conditions in which the agent is to switch between each one. They were primarily developed to be used within video games as an intuitive way for designers to work on non-player characters' AI[35] and can be thought of as a more modern alternative to FSMs. They have however grown in popularity and have over time begun to see use in robotics[36], particularly for programmers with less experience as they are easier to use[37]. Each individual behaviour in a BT is represented by a function which can be compared to a re-usable subroutine in in traditional programming, meaning this allows the agent to intrinsically remember where they are in the structure once they finish the task, rather than being limited to only knowing what state they are in needing to be specifically told where to go by the state itself, where each transition is more akin to a Goto statement. This makes BTs significantly more modular[38] as each task can be more easily placed within a tree as the structure does not require each task to know anything about their neighbouring nodes; instead, the tree itself takes care of coordinating everything and each node only needs to worry about itself. This becomes a greater and greater advantage as the size of the structure increases; very complex agents with a large number of possible states suffer from very poor

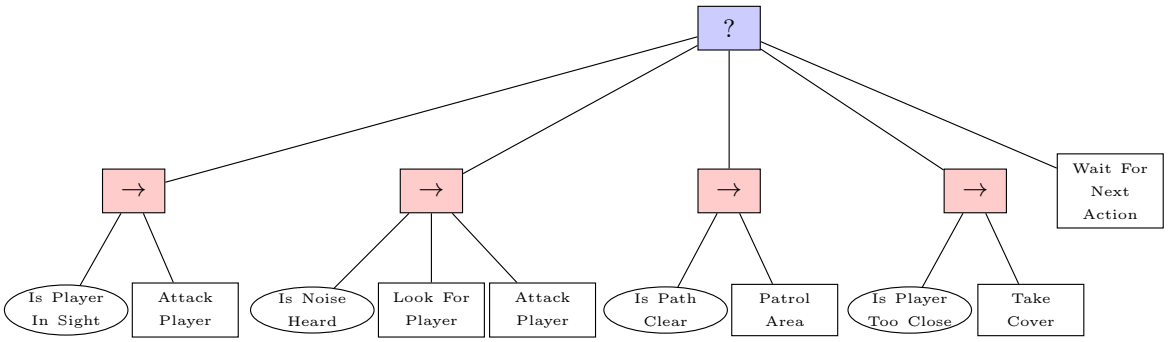

Figure 2.4 : Behavior tree of a typical soldier enemy in Call of Duty.

modularity if there exist links between every single state, however reducing this coupling leads to poor reactivity by the agent, making it worse at properly adapting to situations. BTs surpass this limitation by using signals called "ticks" which rapidly and repetitively traverse the entire tree depth-first, left to right. Every new node encountered contains some information on where to go or what to do, and once the tick has performed whatever needed to be done at a node, it returns upwards to the node's parent carrying a value indicating whether the operation was a Success, a Failure, or is still in progress (Running), which can then be used by other nodes to make decisions[39]. There are two major types of nodes that can be used in a BT: Execution nodes and Control flow nodes. Each type has a number of subdivisions which are explained in the next sections and a summary of which can be found in table 2.1. Figure 2.4 shows an example of a BT for a typical *Call of Duty* enemy soldier who shoots at players on sight, takes cover when players get too close, and otherwise patrols and investigates noises.

2.4.1 EXECUTION NODES

Execution nodes represent behaviours on a granular level. They simply execute some form of code and then return a value; they cannot have children nodes under them. They can either be Action nodes or Condition nodes. Action nodes are simple: when a tick reaches an

Action node, the agent simply performs its associated action by running the code corresponding to it. This may be simple, nuclear actions such as setting a variable in the code, or be something more complex like performing a series of attacks. It is up to the designer how explicit the tree should be in describing tasks. After the action begins, the tick immediately returns to its parent node with the value "Running" and then continues to traverse the rest of the tree based on what the parent node tells it to do. Once a new tick arrives at the Action node, if the action is still in progress, it will also return Running, however if the action is done, then it will return Success or Failure, depending on if the action was successful or not. The Action node is visually represented by a rectangular node with the name of the action inside it.

The other type of execution node is the Condition node. It behaves identically to an Action node, except that there is no tangible action being done, and instead it serves to perform a logical check, which generally takes no time to execute and therefore should never return Running, though I suppose it is theoretically possible if it is a very expensive check, depending on implementation. If the check returns True, then the tick returns a Success, otherwise it returns a Failure. Both of these node subtypes are rather simple, and most of the intricacies of how BTs work lie in the next type. The Condition node is visually depicted as either a box with rounded corners or an ellipse with the name of the condition inside it.

In the BT for the *Call of Duty* soldier in Figure 2.4, the execution nodes are the leaves in rectangle nodes, such as the ones which read "Attack Player", "Look For Player", etc. The condition nodes are the leaves in ellipses, i.e. the ones which read "Is Player In Sight", "Is Noise Heard", etc.

2.4.2 CONTROL FLOW NODES

Control flow nodes are the nodes responsible for directing ticks to where they need to go. Unlike Execution nodes which cannot have children, Control flow nodes must have children, as they do nothing on their own; their only purpose is to manipulate the flow of ticks or the data they carry. There are four types of Control flow nodes: Fallback (also known as Selector[40]), Sequence, Parallel, and Decorator, with the first two being the most commonly used.

The Fallback node instructs the tick to visit and execute each of its children in a given order (by default, left to right) until one of them returns a Success or Running, and then returns that value back to its own parent, including a Failure if all its children returned so. Its purpose is to only perform one action out of a selection; it can be conceptually thought of as an "OR" node. There exist variations of the Fallback node where children are visited in different orders; for instance the Stochastic node is a Fallback node which tracks how often each of its children has succeeded and failed, and over time reorders them to attempt the more successful ones first. There are deeper variations within this as well, for example, a Stochastic node may be set to track separate success rates in various conditions, like a robot whose task is to walk across different types but isn't very good at doing so on icy terrain may want to change its priorities when icy terrain is detected[41]. There is also the similar Utility node, which evaluates the average benefit of each child node through a heuristic algorithm, and then attempts them in order from best to worst. Lastly, there are also Hint nodes which allow the user to directly modify the priority on each child in real time[39]. The Fallback node is indicated by a square node with the symbol ? in it.

Next, the Sequence node acts similarly but instead continues until a child returns a Failure or Running, at which point it returns the same; its goal is therefore to execute a entire

sequence of actions if possible, and only stops if it becomes impossible to do the entirety of it. It can thus be seen as the "AND" counterpart to the Fallback node, as it only ever returns Success if every one of its children was successful. It is visually indicated by a square node with the label $\rightarrow$ in it.

The Parallel node, less frequently used, acts somewhat like a Sequence node where only some of the children must succeed rather than all of them. It must be assigned an integer value M, which represents the threshold of how many successful children are required for the node to then itself return a Success to its parent. It works by firing off every single one of its N children simultaneously and then returning Running until it receives back either M Successes or $N-M+1$ Failures, which is the point where M Successes would be impossible, and then returns the appropriate value respectively. The Parallel node is identified by a square node with the symbol $\rightrightarrows$ in it.

Last is the Decorator node, which is unique in only being able to have one child. For this reason, it is also sometimes considered its entirely distinct type of node rather than a subtype of Control flow nodes. Its role is to apply various user-defined modifications to the return value of its child. It may, for instance, invert its child's return value from Success to Failure and vice versa, or it may be set to only check its child once and forever return the value of that first check when checked again subsequently. In the latter case, it would optimally not even execute the child at all, which is why it can still be considered a Control flow node. Its symbol is a rhombus ($\diamondsuit$).

The *Call of Duty* soldier BT in Figure 2.4 has a Fallback node as its root in blue, and four Sequence nodes under it, in red. Table 2.1 summarizes all the possible nodes found in BTs, their symbols, and when they output each possible tick value.

Node type	Symbol	Success	Failure	Running
Action	[]	On successful completion	If cannot complete	During action
Condition	O	If true	If false	Never
Fallback	?	Once a child succeeds	If all children fail	While a child is Running
Sequence	$\rightarrow$	If all children succeed	Once a child fails	While a child is Running
Parallel	$\rightrightarrows$	Once M children succeed	Once $N-M$ children fail	If neither Success nor Failure
Decorator	$\diamondsuit$	User-defined	User-defined	User-defined

Table 2.1 : A summary of the visual representation and output values for each of the various node types which can be found in Behaviour Trees

CHAPTER III

RELATED WORK

In order to understand where our research stands in the current scientific literature, this chapter presents a summary of the relevant work done on this subject in recent years.

3.1 RESEARCH QUESTION

The literature detailed in this chapter was examined with the goal of answering the following research question: "How can behaviour trees (BT) be used alongside or as an alternative to finite state machines for the purpose of automating the assessment of the comprehensive difficulty of video game enemies?" The key aspects of this question are as follows: first, the assessment of comprehensive difficulty must be touched upon in some way. Targeted articles are those which offer insight on how one might determine comprehensive difficulty such as formal metrics or testing showing a correlation between it and some other element. In other words, the articles should ideally distinguish between difficulty types and not broadly tackle difficulty as a single concept. Second, the work detailed within must be suitable for an automated process early on in the development phase of a game, meaning player testing is off limits outside of offering insight on potential formal game-centric metrics. This means that articles which focus on player-centric metrics such as the rate at which players are able to beat a level or the damage they take or dish out are not particularly relevant. Likewise, Machine-Learning algorithms which require player data to function are also unsuitable. Lastly, the current focus of this master's thesis is to fully understand BT-based solutions and metrics. As such, any article related to them or graphs in general are of great interest, though due to their sparsity, articles matching the other two key aspects are thus still reviewed.

3.2 SEARCH STRATEGY

Articles were obtained through a mix of searching Google Scholar and Academic Search Complete, as well as occasionally diving deeper into the references used in relevant articles. The AI tool Elicit was also used to find connected work. A range of keywords were used, initially fairly general in order to gauge the volume of work in the field of video game difficulty analysis, and then more strictly specific to target comprehensive difficulty as well as formal metrics usable in automated solutions. The keywords were: video game, difficulty, evaluation, estimation, measure, metrics, complexity, comprehensive difficulty, behaviour tree, and graph. The end result consisted largely of hits from the more generic searches which were then handpicked based on relevancy, whereas the more specific searches yielded relatively little of interest.

3.3 RESEARCH REVIEW

The evaluation of video game difficulty has been a budding subject in the last few decades[42, 43, 44], however, many of these are based on testing players and therefore present two significant issues. First, they require a large amount of data that can only be obtained from live testing on a finished or nearly finished product. As mentioned previously, playtesting is a heavy time sink in the development process of a game, one which could theoretically be done in parallel with the rest of development if it weren't for the fact that it can only happen near the end of it. Second, due to how most of the data is extracted from player performance metrics such as score, success rate, speed, etc., the assessed difficulty is purely player-centric, which is a subjective form of difficulty which can vary significantly from player to player based on their skill and experience with a game. This type of data is appropriate for real time adjustments such as seen in Dynamic Difficulty Adjustment (DDA) practices, but is less desirable for automatic measurements. Researchers have only more recently started putting

more effort towards finding automated solutions. Still though, this early foray into the subject has nonetheless laid some of the groundwork for determining what may be potential metrics in future work, such as enemy count in a First-person shooter game[45].

Unfortunately, research involving manual testing is not the only type which must be filtered out for this project, as a very large percentage of all research on video game difficulty focuses on executive difficulty[46, 47, 48, 49, 50, 51], meaning it analyzes how demanding games are with regards to the player's ability to control their character, aim at enemies, or perform difficult controller inputs. An example of this is one of the recent articles by Francillette et al.[52] which proposes the use of virtual pheromone trails to form a heat map of the most dangerous areas of a level in a platforming game, which aims to provide swift feedback to designers in order to make adjustments depending on their vision of a target difficulty. There is also a number of articles which don't isolate a particular type of difficulty at all, choosing instead to broadly analyze difficulty as a whole[53, 54, 20, 19]. Relatively few articles mention comprehensive difficulty at all[13], and fewer still focus on it entirely. Once again, a recent article by Francillette et al. is one of those few; in it, they suggest the use of cyclomatic complexity as a formal design metric for automatic analysis of enemy difficulty as a result of their behaviour[55]. This is in fact the article which primarily introduced me to the concepts which serve as a basis for what this master's thesis builds upon.

When it comes to automated modelisation of video game difficulty, there is a wide range of different methods that have been explored by various teams. Some propose the use of AI agents to play the game like they are real players, which enables designers to gather data on specific game balance aspects such as statistics on the average damage output of a player or how often certain events occur, etc. Interestingly, in his doctoral dissertation, Jaffe[56] cites an article by Hom and Marks[57] where an AI agent was used to aid in the balancing of board games, which are strategy games all the same as computer strategy games. Jaffe however raises

the issue that this kind of data collection can be skewed for a variety of reasons, such as the AI being biased towards a certain strategy which employs specific moves disproportionately often, which could be mistaken as meaning that those moves are too strong even though other strategies could very well employ completely different moves. He proposes a solution to this in the form of "restricted play", which is a more objective method of determining the impact of a specific move or mechanic in the balance of a game by matching two identical AI agents against each other but disallowing one of them from using the mechanic being studied and observing the difference in success rate.

3.3.1 ENEMY EVALUATION AI FOR 2D ACTION-PLATFORM GAME

Likewise, Promsutipong et al.[58] attempt a similar technique, but for video games rather than board games, by combining the use of a FSM, search algorithms and rule-based heuristics to develop AI agents which play like human players. This is intended to help developers test changes to their game by simulating large batches of human-like game-play sessions without relying on actual human testers. The experiment took place in a simple 2D platformer where the goal was for players to navigate a level from left to right while avoiding or fighting enemies throughout. As mentioned, the AI uses a FSM with two states, fighting and wandering, which imitates how human testers played, which was to attempt to avoid encounters with enemies until they were spotted and then they would to trying to deal damage. In the fighting state, the AI searches for the best button combination with the lowest cost to move its character in order to avoid hits, and would simply always attack whenever it was facing the opponent and was in range of it. In the wandering state, it would use simulation of its own movement in conjunction with regression analysis of enemy movement to attempt to path around enemies without drawing their attention. In the event of a tie between multiple different options in the decision-making process, heuristic rules were used to pick a strategy

between Stand, Escape or Fallback. If the AI's character detected an enemy from far away, was on the floor and not currently using the Escape strategy, the AI would choose the Stand strategy. If the previous was true with the exception that the enemy was nearby, the AI would choose the Escape strategy. Lastly, if any of the previous conditions were false, it would choose the Fallback strategy. This allows the AI to behave consistently but also to mimic a human's state of hesitation and situational decision-making.

The main drawback of this method is that while this does eliminate the need for human testers in repeated testing of an established game, it still requires a dataset as a base, which can only be obtained through human play-testing, thus not allowing for analysis early in the development cycle. However, one interesting aspect of this approach is that it abstracts the physical aspect of video games, as AI do not need to physically push buttons in order to play, allowing us to better isolate the comprehensive difficulty of a high intensity game away from the motor difficulty.

3.3.2 USING APPLIED COGNITIVE LOAD THEORY AND DIFFICULTY ANALYSIS FOR EDUCATIONAL GAME DESIGN FOR UNDERSTANDING AND TRANSFERENCE OF LITERACY SKILLS IN ADULTS

In this article, Ouellette et al.[53] detail how they used Cognitive Load Theory to analyze the gameplay curve of Codex: The Lost Words of Atlantis, a serious game aimed at improving literacy in adults, in an attempt to bring a new approach more rooted in empirical data to designing educational games. Cognitive Load Theory dictates that there are three types of cognitive loads: intrinsic load, which is caused by the inherent, real difficulty of a task; extraneous load, which is added difficulty due to distractions, context or any other obstacles that get in the way of learning; and germane load, which is the working memory's capacity for processing new information and linking it to existing information found in long

term memory. While it would be interesting to tackle cognitive load as a whole, our project currently limits its scope to real, objective complexity, which in this case is intrinsic load. In order to determine the intrinsic load of various tasks, the authors of this article cite the English as a Second Language National Reporting System (ESL NRS) chart by the authors of the Comprehensive Adult Student Assessment System (CASAS)[59] as a base, defining the respective difficulty of each task and then increasing it by a multiplier based on various situation modifiers representing extraneous load, such as for instance, the challenge having a time constraint. The goal of the experiment was to plot the game's difficulty curve and to attempt to smoothen it, should it not fit a desired flow curve. Results indicated that the initial curve featured spikes in difficulty which were removed after their source was identified; a challenge where players must match words together. This "form-fill" challenge featured words that were more complex than those of other challenges to compensate for the questions being essentially multiple-choice questions. Following the replacement of this challenge with a more standard challenge, the difficulty curve became smoother and more accessible. This further demonstrates that the more analytical approaches to game design that our project seeks to bring forth are indeed possible.

This specific example is all the more interesting because literacy can be seen as a form of comprehensive difficulty. However, I was not able to access the chart used in the experiment, and the article does not explain how the values for the difficulty ratings were obtained in sufficient detail, limiting its usefulness right now. It is certainly possible that deeper digging into the metrics used to construct that chart could prove insightful.

3.3.3 HOW HARD IS IT REALLY? ASSESSING GAME-TASK DIFFICULTY THROUGH REAL-TIME MEASURES OF PERFORMANCE AND COGNITIVE LOAD

Similarly, Seyderhelm and Blackmore explain in this article[60] an experiment in which they made participants play a driving game in different environments carrying varying degrees of visual noise, while also sometimes performing secondary tasks alongside driving, such as counting other vehicles of a particular colour. The objective of this experiment was to study the effects of gameplay challenges and different environments on players' cognitive burden and task performance. They measured the 31 participants' cognitive load in accordance with ISO 17488[61] as they played through a series of ten zones composed of three similar levels each. Each zone featured a different environment and asked participants to complete a primary driving task as well as a miscellaneous secondary task in some instances. The metrics used to measure cognitive load included virDRT response times and a combined metric known as the Inverse Efficiency Score (IES)[62]. Findings showed that cognitive load was not directly proportional to task difficulty. For instance, the fourth and tenth zones both involved difficult gameplay tasks but no secondary task, yet participants experienced fairly low cognitive load on zone four, possibly indicating they had reached a flow state, but struggled on level ten which featured the more complex task out of the two levels, resulting in high load and poor performance. Moreover, level two, which did not feature very difficult gameplay but did require participants to listen to, memorize, and apply verbal driving directions, showed the highest difficulty score out of the ten zones.

It is my belief that cognitive load is a viable metric for gauging complexity and therefore comprehensive difficulty, and that such charts and standards could provide insight on how it could translate to traditional video games. Unfortunately, as it stands, these methods are mostly intended for DDA and are therefore ill suited for our project.

3.3.4 DIRECTLY CONTROLLING THE PERCEIVED DIFICULTY OF A SHOOTING GAME BY THE ADDITION OF FAKE ENEMY BULLETS

Tangentially to the topic of cognitive load, Zhang published a study[63] on the effects of fake challenging visuals, such as illusory bullets in a shooting game, on perceived difficulty. Eighteen participants played a simple top-down 2D shooting game through four timed trials where they must shoot a series of randomly placed targets while avoiding being hit themselves by enemies. All four tests contained the same amount of real enemies shooting the same amount of real bullets. In the first and third test, the game operated normally, with enemies shooting only real bullets at the players. In the second test, additional enemies were spawned which would only ever aim away from players, at a distance calculated to be just barely far enough away to guarantee players could not get hit even if they ran towards the bullets; referred to as "unreachable bullets" by the authors. In the fourth test, additional enemies would instead shoot at the players, but their bullets would not deal damage to the players and would instead phase through them harmlessly. Results found that the participants found no significant increase in perceived difficulty in the second test, but that the fourth test did significantly increase it.

It is not surprising that the addition of perceived threats aimed at the players significantly raised their sense of difficulty despite not actually affecting how dangerous enemies were, however it is unexpected that the increase in the number of on-screen bullets that couldn't reach the players did not have a major impact. This once again provides insight into the psychological factors which may affect a player's ability to gauge the challenge of a situation, and therefore their understanding of it. However, in terms of Cognitive Load Theory, these additional bullets are more akin to extraneous load, meaning it isn't a measurement of the real, fundamental complexity of a system but rather an overhead added onto it.

3.3.5 ILLUMINATING THE SPACE OF ENEMIES THROUGH MAP-ELITES

Viana et al.[64] propose an inventive approach to the task of designing enemies. They have put together a method for procedurally generating enemies through the use of a MAP-Elites algorithm[65], which is a type of optimization algorithm which specifically avoids falling into the pitfall of local optima by acknowledging that a solution may be best in a certain context but that other paths may be better suited for other contexts. The algorithm is given a design space composed of various enemy attributes common in Action-Adventure games, such as health, attack damage and speed, movement speed and type, and weapon type. It then explores that design space and attempts to generate enemies "optimized" according to difficulty constraints by iterating and converging towards the target difficulty. In other words, enemies intended to be easy had a lower attribute pool total than their harder counterparts. The algorithm proved to be quite efficient, with most enemies being generated within 160 milliseconds, and showed improved quality and diversity compared to a Parallel Evolutionary Algorithm (PEA) previously used by the team. Next, a prototype Action-Adventure game was developed, and the procedurally generated enemies were implemented within. The game was then tested by 96 participants over 124 levels featuring enemies of diverse difficulty. The participants successfully found the enemies to match expected levels of difficulty, many even stating they believed the enemies to be designed by hand, especially in the more difficult levels, suggesting the system was successful in producing convincing enemy design.

While the automated content generation goes beyond our scope, the use of enemy attributes in the process is once again a promising lead on the identification of formal metrics of enemy difficulty, though at the current stage more work yet needs to be done to explore that direction.

3.3.6 PROCEDURAL LEVEL GENERATION WITH DIFFICULTY LEVEL ESTIMATION FOR PUZZLE GAMES

This article by Spierewka et al.[66] displays the use of procedural level generation in the puzzle game inbento[2], shown in Figure 3.1, where players are tasked with fulfilling food orders of japanese lunch boxes called "bentos". The objective is to perfectly match a desired layout for the food in the form of "tiles" of various types of food by placing pre-made food blocks reminiscent of tetris pieces, with an important distinction being that blocks can also be placed over tiles which already contain something to overwrite them. The bentos vary in size from 1 by 2 all the way to 3 by 4, with later levels also adding special blocks which have various effects on the current state of the lunchbox rather than adding food tiles, such as swapping two tiles' positions, copying existing tiles, etc., thus adding even more complexity. In the article, both an algorithm for evaluating a level's difficulty as well as an algorithm for generating levels were designed. The former simply summed various weighted values based on variables such as the size of the level, the number of pieces and the number of different food types involved. It was first tested on the game's existing base levels and the authors found that it accurately depicted the game's estimated difficulty curve. The generation algorithm used a Breadth-First Search (BFS) to explore every possible placement of a given set of pre-chosen pieces. If at any point during exploration, it generated a state that had already been generated previously, that state was discarded, thus ensuring that the final result required every available piece; as otherwise, it could naively generate a solution where all the blocks are placed on the same tile, overwriting each other and leaving only one of them as actually required. Duplicates found only on the final depth were kept but were identified as such, with the premise that levels with multiple paths to the same solution were valid but considered easier. The algorithm was then used to generate a number of levels of incrementally increasing difficulty, which were

[2]https://store.steampowered.com/app/1567440/inbento/

Figure 3.1 : The Steam store page for inbento

then tested by the evaluating algorithms, and results found that the difficulty curve matched the game's base levels.

This article is noteworthy for being about a puzzle game's general difficulty, yet its metrics closely resemble those of cognitive load and comprehensive difficulty, highlighting the occasional overlap between difficulty types in some game genres. One slight difference in this one compared to articles reviewed earlier is that the difficulty equation uses a sum of weighted values rather than a product. It ultimately produces results that are nonetheless mostly proportional to real difficulty, which is expected on such a small scale. Unfortunately, this article's findings are unique to puzzle games, and would not apply to other genres of games.

3.3.7 SUMMARY

There are multiple publications aiming to analyze video games in various ways, but none specifically match the way we are investigating. Many of them are simply not considering comprehensive difficulty at all, and many of those also focus on methods that can only be applied later in a game's lifespan, whether by automatic adjustment based on player performance or through training an AI to test the game based on player data. Once again, the common problem here is the reliance on players to generate data before any proposed method can be applied. Some articles nonetheless provide useful insight on potential directions for future work, but these are not presently relevant in the scope of this master's thesis.

CHAPTER IV

MATHEMATICAL MODEL

The model proposed in this master's thesis aims to define a formal metric which can be used for both FSMs and BTs and allows them to be seamlessly compared with each other. First, I define the metric used to measure the comprehensive difficulty of enemies. Then, I provide algorithms showing how an enemy's representation can be converted back and forth between the FSM form and the BT form. Lastly, I prove exactly how the metric scales between both forms, allowing for direct comparison between FSM and BT without needing to convert the structure itself. Please note that for the purpose of keeping things concise in demonstrating this proof of concept, the Parallel and Decorator BT nodes mentioned in subsection 2.4.2 are not considered.

4.1 METRIC DEFINITION

This thesis uses the same definition for comprehensive difficulty as the recent article by Francillette et al. on the topic[55], which is the product of the volume of information contained in a graph G and its cyclomatic complexity. As explained previously, the cyclomatic complexity of a graph is a metric detailing the complexity of its structure, which is largely a factor of the coupling between its vertices; a graph with a large number of edges is more complex than another graph with the same number of vertices but fewer edges. Inversely, a graph with a similar number of edges but a greater number of vertices is actually less complex despite containing more information, as more of that information is part of the same independent paths within the graph, typically making it relatively easier to understand. The equation for cyclomatic complexity used here is taken from McCabe's original article on

it[34] and is shown in Equation(4.1), with m is the cycle rank of the underlying graph, i.e. its cyclomatic complexity, e is the number of edges and n is the number of vertices:

$$m = (e - n + 1). \tag{4.1}$$

Of course, that alone is not sufficient to characterize comprehensive difficulty as a whole; in the end, larger graphs with more information still require more cognitive effort to understand, thus justifying the inclusion of the volume of information in the product. The volume of information is simply defined here as n, the number of vertices in the graph. This means that formally, the full equation for the comprehensive difficulty c is as follows:

$$c = mn = n(e - n + 1) = n(e + 1) - n^2. \tag{4.2}$$

It is worth noting that as [55] mentions, this equation is intended for use within a strongly connected graph. In order to meet that requirement, we must add a fictional edge from the final node to the starting node. When considering a BT, it has not been properly defined where one or more fictional edge should be added; in my master's thesis, I've chosen to add them after every action on the deepest level of the tree, linking back to the root node. The reasoning for is that those are the only nodes in the tree where a tick's execution thread can possibly end, ignoring edge cases involving Running values.

To showcase this, let us look at some examples of video game bosses, some of which have few different moves but use them in complex ways, whereas others have a large variety of attacks but are fairly easy to understand. Consider a very simple boss such as Whomp King in *Super Mario 64*; he only has one attack, and does not vary how he uses it whatsoever. In a spectrum defined by the volume of information and structure complexity, Whomp King

would be at the bottom of both. Consider instead a boss who has many different moves but remains relatively simple, such as Master Hand[3] in *Super Smash Bros*. Master Hand has 11 different attacks but uses them in very simple ways. It uses them one by one and waits a few seconds between each one, never chains them together, chooses them entirely randomly with no specific condition triggering any one of them, always targets the player's immediate position with them, and only one of them even has a variation whatsoever, where it becomes a triple hit when at low health. In the aforementioned spectrum, it would be high on the information volume scale but remains low on the complexity scale. In the opposite corner, an enemy with few moves but a high degree of complexity between them would be Mike Tyson[4] from the original *Punch-Out!!* for the Nintendo Entertainment System. He only uses five different attacks, but he chains them together in patterns that are difficult to predict. So far, only enemies from older titles have been named, partially because they are easier to analyze. In comparison, the average enemy from most modern titles eclipses them in complexity. Still, to use a more extreme example, one could look at the Dancer of the Boreal Valley[5] from *Dark Souls III*. Notoriously one of the hardest bosses in the game, her arsenal features a staggering 34 different attacks with various conditions determining their use, as well as multiple of which may be chained together. Motor difficulty is definitely not the only challenge here, and mastering this boss requires players to spend a great deal of time studying her mechanics.

4.2 CONVERSION BETWEEN FSM AND BT

In the interest of keeping comparisons as objective as possible between enemies from different games, who may have their behaviour structures designed with vastly different philosophies, the first step is to designate a form as the canonical form. This form is obtained

[3]https://www.ssbwiki.com/Master_Hand_(SSB)

[4]https://punchout.fandom.com/wiki/Mike_Tyson

[5]https://darksouls3.wiki.fextralife.com/Dancer+of+the+Boreal+Valley

by converting existing BTs into FSMs and then back into BTs. The algorithms are deterministic and result in BTs which always follow the same structural morphology. For enemies represented as FSMs, a direct conversion to BT suffices.

Algorithm 1 is a direct translation into pseudocode of the conversion concept introduced in the book *Behavior trees in robotics and AI: An introduction*[39]. It converts a BT into a FSM by converting each node by starting from the root and recursively working down each child. All nodes (T) become a subgraph made up of five vertices; one entry state (IN), one middle node representing the node itself (G), and lastly, three exit nodes representing the outcomes of a success (S), a running state (R), and a failure (F), respectively. The entry node always feeds into the middle node, which then always feeds into the remaining three nodes, and then the middle node recursively becomes a five vertex subgraph of its own as there is another level under it, with the final level always being an execution node by definition, meaning in the final result, the middle node of every subgraph is always a state representing an execution node (hereby referred to as G_N). However, how all the layers of entry nodes and output nodes surrounding each execution state connect to the rest of the FSM is determined by what type of control flow node its parent (now T or G, depending on whether we are referring to its BT form or its FSM form) was and whether the execution node was its parent's rightmost child or not (in other words, whether a G_{N+1} exists). If it is the rightmost child, all three of the child's S, R, and F nodes connect directly to its parent's respective output nodes. Otherwise, R always connects to its parent's R node regardless, but if the parent is a fallback node, then a failure means the control flow must continue to test G's remaining children, whereas a success means the control flow is done here and must return to the previous level. Therefore the algorithm connects G_N's S node to its parent's S node and G_N's F node to its next sibling's IN node. Inversely, if the parent is a sequence node, then the opposite connections are made, with S going to its next sibling and F going back to the parent's output.

```
Require: a behavior tree T
Ensure: an equivalent Finite-State Machine
  function MAIN(T)
      set q_0 as a state representing the tick source
      BTTOFSM(T)
      connect S, R, F to q_0
  end function

  function BTTOFSM(T)
      create new subgraph G
      set IN as the entry state into G
      set S, R, F as G's exit states
      for all children T_N of T do
          set G_N = BTTOFSM(T_N)
          if T_N is not the last child of T then
              if T is a fallback node then
                  connect G_N's S and R nodes to G's S and R nodes
                  connect G_N's F node to G_{N+1}'s IN node
              else if T is a sequence node then
                  connect G_N's F and R nodes to G's F and R nodes
                  connect G_N's S node to G_{N+1}'s IN node
              end if
          else
              connect G_N's S, R, and F nodes to G's S, R, F nodes
          end if
      end for
  end function
```

Algorithm 1 : BT to FSM conversion

Enemies in video games do not require FSMs that track intermediary states such as *IN* or *S*, *R*, and *F*. In typical FSMs for video game enemies, those states are represented directly by the transitions between the real states. The transition states essentially only exist for the purpose of keeping the algorithm lighter, as otherwise only execution nodes would need to connect to other nodes, leaving a relatively heavy task of tracing back which other execution node it should connect to based on an indefinite number of parents, which would be difficult to

represent in pseudocode. It is simply easier to "remember" where the control flow should go by creating these intermediary states that child nodes can easily connect to without worrying about where they output to. For this reason, I propose converting all of these intermediary states into typical FSM transitions labelled accordingly. Moreover, because the "running" response in BTs always causes the end of the tick's execution thread, it can never result in a change of state, I propose removing all "running" transitions. Another way to think of it would be if all "running" transitions simply looped back into the state they originated from, since a real transition won't actually happen until the state decides whether it has succeeded or failed and making the state continuously loop into itself would mimic how BTs continue to send ticks to the same node until it is no longer running; however adding this to the graph would add unnecessary bloat, so I've opted to simply remove them entirely.

Algorithm 3 reverses the conversion by turning each state into a subtree consisting of a sequence composed of a condition node which first checks if the enemy is in the correct state, an action node which performs the action tied to the state, and then another fallback subtree which iterates through each possible transition from that state in order to see if any of them must be performed. That last part is done by a series of two node sequences with a condition checking if the transition is to be triggered, and an action which changes the state variable. Altogether, all the sequence subtrees generated by each state node are then grouped under a root fallback node, to be ticked one by one until the one corresponding to the enemy's current state is found. This makes for a rather bulky but ultimately quite simple and linear BT; it can be summarized by three checks: find the state the enemy is in, then perform its action, then find out if a transition needs to be done; repeat ad nauseam.

Require: a behavior tree T
Ensure: an equivalent Finite-State Machine
  function MAIN(T)
    set q_0 as a state representing the tick source
    BTTOFSM(T)
    connect S, R, F to q_0
    replace every IN, S, R, and F node with a direction transition between their two connected neighbours
  end function

  function BTTOFSM(T)
    create new subgraph G
    set IN as the entry state into G
    set S, R, F as G's exit states
    for all children T_N of T **do**
      set G_N = BTTOFSM(T_N)
      if T_N is not the last child of T **then**
        if T is a fallback node **then**
          connect G_N's S and R nodes to G's S and R nodes
          connect G_N's F node to G_{N+1}'s IN node
        else if T is a sequence node **then**
          connect G_N's F and R nodes to G's F and R nodes
          connect G_N's S node to G_{N+1}'s IN node
        end if
      else
        connect G_N's S, R, and F nodes to G's S, R, F nodes
      end if
    end for
  end function

Algorithm 2 : Simplified BT to FSM conversion

Require: a Finite-State Machine M
Ensure: an equivalent behaviour tree
  create *root* as a new fallback node
  set $i = 0$
  for all states S in M **do**
    create a sequence node *Seq* under (as a child of) *root*
    create a condition node under *Seq* labeled "$State = S$"
    create an action node under *Seq* which performs actions corresponding to state S
    create a fallback node Fb under *Seq*
    for all transitions T starting in S **do**
      create a sequence node $TransitionSeq$ under Fb
      create a condition node under $TransitionSeq$ representing the condition to achieve that transition
      create an action node under $TransitionSeq$ which sets $State$ to a new value matching the transition
    end for
  end for

Algorithm 3 : FSM to BT conversion

4.3 MATHEMATICAL JUSTIFICATION

In order for comparison between the two models to be valid, we must ensure they produce metrics that are asymptotically equivalent to one another. Let us look at three propositions to this effect:

Theorem 4.3.1. *Given a BT with n nodes, l leaves and e edges, applying Algorithm 1 to it results in a FSM with* $1+4n+l$ *nodes and* $4n+3l$ *edges.*

Proof. First, the root is turned into a subgraph with 5 vertices, of which the middle vertex is transformed into as many 5 vertex subgraphs as it had children. This is repeated n times, for a net increase of 4 vertices per node originally on the BT. Since the nodes on the last level of the tree retain their middle nodes after being converted, the total adds up to $1+4n+l$ vertices when including the q_0 node added afterwards. The majority of nodes in the resulting FSM

only have one outgoing transition; only the middle nodes from each original leaf have 2 more than the rest for a total of 3, and therefore the total number of edges is $(1+4n+l)+(2l)$, or $1+4n+3l$. ■

Theorem 4.3.2. *Given a BT with n nodes, l leaves and e edges, applying Algorithm 2 to it results in a FSM with* $l+1$ *nodes and up to* $2l$ *edges.*

Proof. By removing the intermediary nodes, we are left with only executive nodes (which are the leaves) as nodes in the FSM, in addition to the q_0 node. Each node can have either a single outgoing transition, or one for a success and one for a failure, except for the q_0 and the final node in the tree which can only have one; this adds up to $2l$ edges. ■

A similar logic ensues for the opposite conversion:

Theorem 4.3.3. *Given a FSM with n nodes and e edges, applying Algorithm 3 to it results in a BT containing* $1+4n+3e$ *nodes and* $4n+3e$ *edges.*

Proof. First, the BT starts with a root node, and for every state in the FSM, we append to it a subtree consisting of four nodes: a sequence node, a condition, an action, and a fallback node. Then, every outgoing transition from each state becomes a three-node subtree under that fallback node, for a total of $1+4n+3e$ nodes. By definition, a tree has $n-1$ edges, and this is the case for a BT. ■

Finally, Corollary 1 follows as a result:

Corollary 1. *Given a BT with n nodes, submitting it through either Algorithms 1 or 2, followed by Algorithm 3 outputs a BT with a node count in the order of* $O(n)$.

This allows us to validate the comparison between the two models and permits us to treat them as equivalent. Let us recall Figure 2.4. Figure 4.1 shows the result of applying Algorithm 2, where the purple transitions are "Success" transitions and the orange transitions are the "Failure" transitions. In summary, the only nodes from the BT that become nodes in the FSM are the execution nodes, which includes condition nodes such as "Is Player In Sight?" and action nodes such as "Attack Player". If we divide the BT into subtrees at each control flow node, we can more clearly see how the transitions are determined in the FSM: if a node is not the last of its siblings, its parent decides whether Success or Failure is the response that connects to its next sibling; Success if the parent is a sequence node and Failure if it is a fallback node. Then, the other response looks at the node's parent's parent to know where it goes, though in most cases it will simply connect to the first execution node of the subtree corresponding to its parent's next sibling. In 4.1, this can be seen with the "Is Player in Sight?" node, whose Success transition connects to its sibling and whose Failure transition connects to the next subtree, in this case the "Is Noise Heard?" node. If an execution node does not have a next sibling, the process is similar, except it directly looks at its parent's parent to know whether to continue on a Success or Failure, and the remaining choice is left to the next parent above it, or simply returns back to the start node if there are no more parents above it, such as is the case with the "Attack Player" node, where its transitions are determined by its parent's parent, a fallback node, meaning its Failure should link to the next node and its Success would be determined by the next parent above it, but instead returns back to the start as there isn't such a parent.

Then, Figure 4.2 shows the final result, the canonical BT obtained from converting the FSM back into a BT with Algorithm 3. Every node in the FSM becomes an action node in the BT, together in a subtree alongside a condition node which checks which state the entity should be in and another subtree which checks every possible transition and whether they should

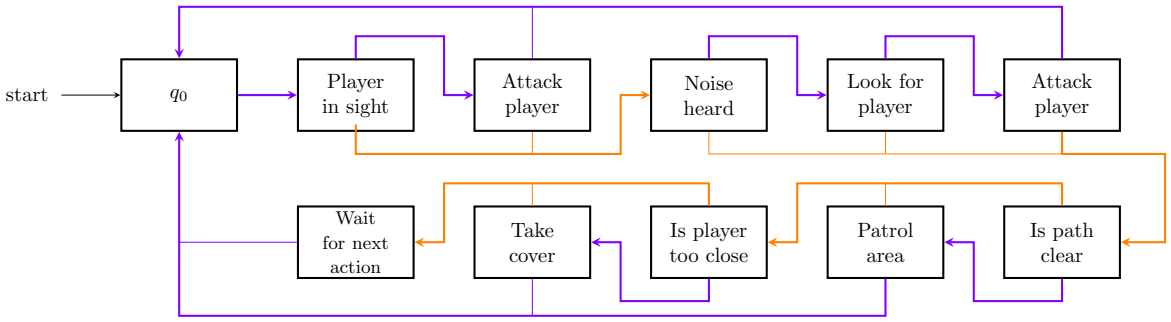

Figure 4.1 : FSM obtained by applying Algorithm 1 to the BT of Figure 2.4.

happen or not, and then sets the state variable accordingly if that is the case. For instance, the "Attack Player" node becomes a subtree which becomes with a condition checking if the entity is in the "Attack Player" state, then executes the "Attack Player" action, then checks if the action was a success or a failure, and sets the state to the start node or to the "Is Noise Heard" action, respectively. The exact same process is repeated for every node in the FSM.

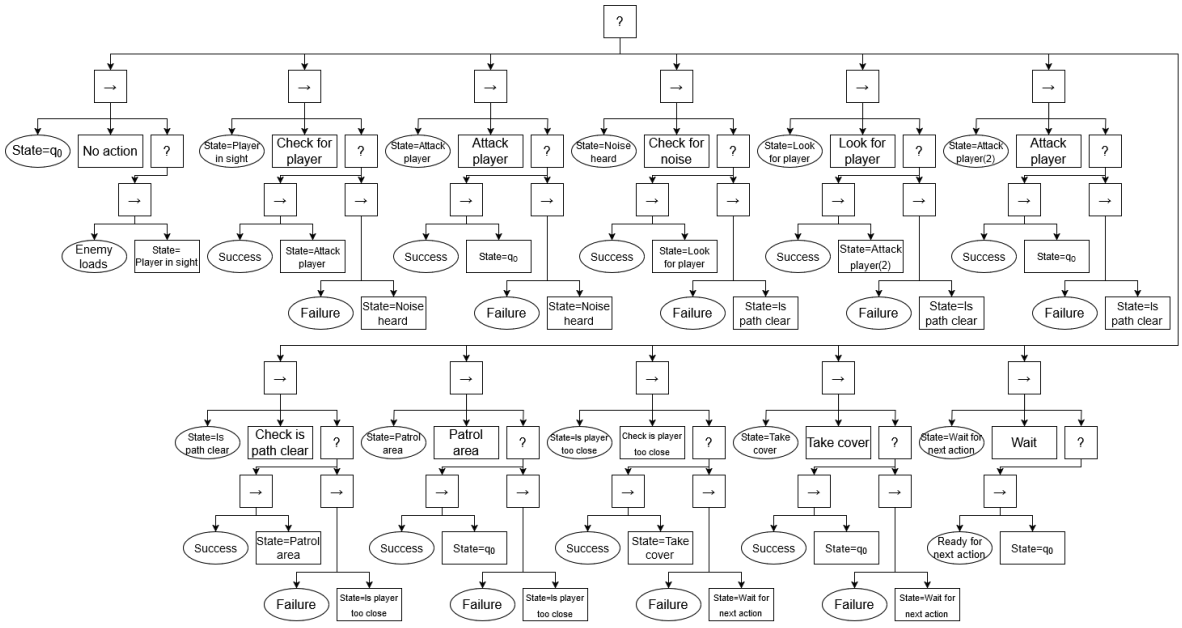

Figure 4.2 : Canonical BT obtained by applying Algorithm 3 to the FSM of Figure 4.1.

CHAPTER V

CASE STUDY AND VALIDATION

As a proof of concept to validate our approach, we selected various enemies from four different games and computed their comprehensive difficulty by means of applying Equation 4.2 to their canonical BTs. The four games we've elected to study are *Super Mario Bros, Mega Man, Super Punch-Out!!* and *Sekiro: Shadows Die Twice*. For the sake of completeness, let us go over a quick summary of each game. The first two are fairly simple platformers released in 1985 and 1987, respectively, wherein the player must traverse 2D levels while avoiding or fighting enemies, obtain power-ups and then eventually arrive at a boss which must be defeated. In *Super Mario Bros.*, the player's main mean of defeating enemies is to jump on them, though one power-up allows the player to shoot fireballs at enemies to deal damage as well. Each enemy displays different movement and attack patterns, from the simple *Goomba* who merely moves left and right to the *Hammer Bros.* who both move and attack erratically and even jump from time to time, making for a very clear curve in difficulty as the player progresses through levels. The player must understand enemies well, as any contact with an enemy or their attacks removes any power-up the player has, and the next hit without a power-up kills them. In *Mega Man*, the combat is slightly more involved: the player deals damage by shooting enemies with their equipped weapons, and has their own health bar which can take a few hits rather than being limited to one or two hits before death like in *Super Mario Bros.*. They begin the game with the generic *Mega Buster*, which can infinitely fire rapid but weak bullets in a straight path. They can, however, obtain new weapons by defeating the boss of each level, which then allows them to exploit different enemies' weaknesses at the cost of limited ammunition and having to master different firing mechanics. The player may attempt any of the first six levels in any order. As each boss features unique attack patterns

which vary in difficulty, this provides an option to fight whichever boss the player finds easiest first while they only have the *Mega Buster* and wait until they have stronger weapons before tackling bosses they find harder.

Next, *Super Punch-Out!!* is a 2D boxing game originally released in 1994 in which the player incarnates *Little Mac*, a low weight class but fast and skilled boxer who must overcome his size disadvantage by outmanoeuvring larger enemy contenders. The controls consist of picking a direction, and then attacking or defending. When attacking, the player may choose to attack from the left or right, aim for the body or the face, as well as choose between a quick but weak attack or a slow but strong attack. Likewise, when defending, the player may choose to block or dodge as well as a direction in which to do so. The player may block low to intercept body blows or block high to protect their face. When dodging, the player may choose to duck or to dodge left or right, though most attacks can be dodged in any direction. Blocking is riskier but allows for counterattacks, whereas dodging is safer but provides no other benefit. Because the game offers no freedom of movement for moving across the ring as a space, the game puts much more emphasis on what the player *can* do, which is read the opponent's actions and react to it. Each enemy boxer features a distinct style and attack pattern as well as an exaggerated personality which usually highlights their specialty; for instance, the second boxer by the player is a 440 pound obese man known as "Bear Hugger". His very large frame serves to indicate that he is a slower but much more defensive opponent than the previous boxer fought by the player, who was rather fragile and served mostly as an introduction to the game where even simple button mashing could lead to a victory. In this second fight, the player must therefore learn that they must analyze their opponent and exploit any openings in their defense.

Last, *Sekiro: Shadows Die Twice*, released in 2019, is what is colloquially known as a souls-like, meaning it shares the same genre as the Dark Souls series. It is a 3D action

role-playing dark fantasy game heavy with fluid combat where the player is expected to expertly dodge or block enemy attacks while keeping an eye out for openings to strike back, all the while managing their stamina as to not leave themselves tired out and exposed. The genre is notorious for even its simplest enemies being deadly, requiring the player to understand them very well despite them often only having enough health to survive one or two attacks. This of course serves as a build-up to the bosses, which are typically even deadlier while having much more health and more complex attack patterns.

Super Mario Bros. was chosen for its simplicity, as it allows for the most direct observation of our approach, leaving little room for interpretation of any complex game rules: the game remains mechanically the same throughout its entirety, all that varies is the complexity of the levels and the enemies the player encounters. On the other hand, *Mega Man*, *Super Punch-Out!!* and *Sekiro: Shadows Die Twice* were chosen thanks to how the goal of understanding enemies' attack patterns takes up a large part of the game-play loop, making these games ideal candidates for our analysis. Other factors, such as release year and diversity of game-play were also factors, overall allowing for a wide coverage upon which to appreciate our approach in terms of robustness, scalability and relevance.

Since we do not have access to the design documentation nor the source code of the games being studied, approximate models were used, in the form of FSM in the case of *Super Mario Bros.* enemies and in the form of BT for the rest. These were first obtained through a ChatGPT prompt in an attempt to be as objective as possible with regards to the granularity level of the action nodes, considering that the granularity level is a choice usually left to the designer's discretion. The models were then adjusted by hand to rectify certain major inconsistencies with the visible behaviour of the modelized enemies; for instance, when ChatGPT suggested a FSM for the Hammer Brothers which contained nodes instructing it to flee from combat, those nodes were removed as a simple observation of their in-game

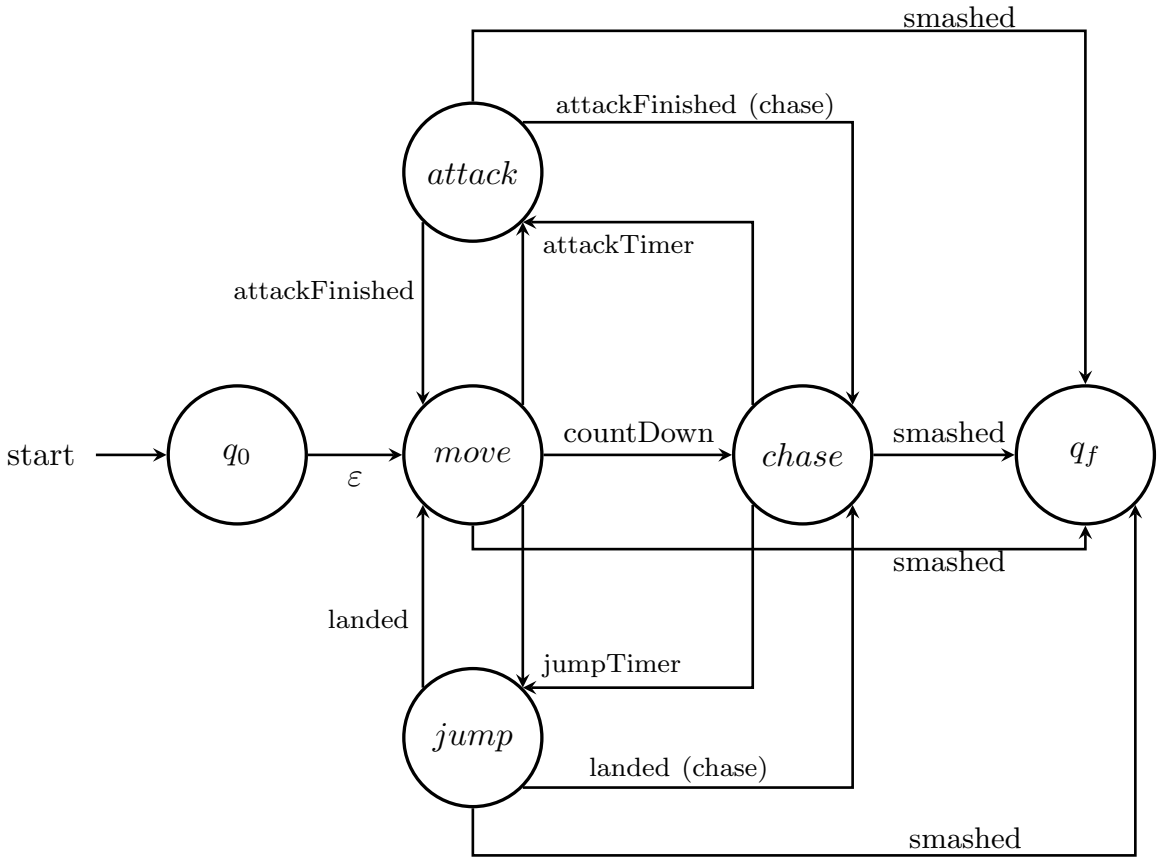

Figure 5.1 : FSM for one of the more difficult enemies in *Super Mario Bros.*, the Hammer Bro.

behaviour clearly shows that they do not ever flee from battle. We then used Algorithm 2 to output a corresponding FSM (except in the case of *Super Mario Bros.* enemies which were already in the FSM form), followed 3 to output a canonical BT for each one. Finally, we measured the comprehensive difficulty for both the canonical BT and the FSM for each enemy and compared them with one another. Figures 5.1 and 5.2 show the FSM and canonical BT for *Super Mario Bros.*' *Hammer Bro.* enemy, whereas Figures 5.3 and 5.4 show the starting BT output by ChatGPT and the canonical BT for *Sekiro: Shadows Die Twice*'s *Ashina Elite* enemy. Table 5.1 shows the list of all the enemies we analyzed.

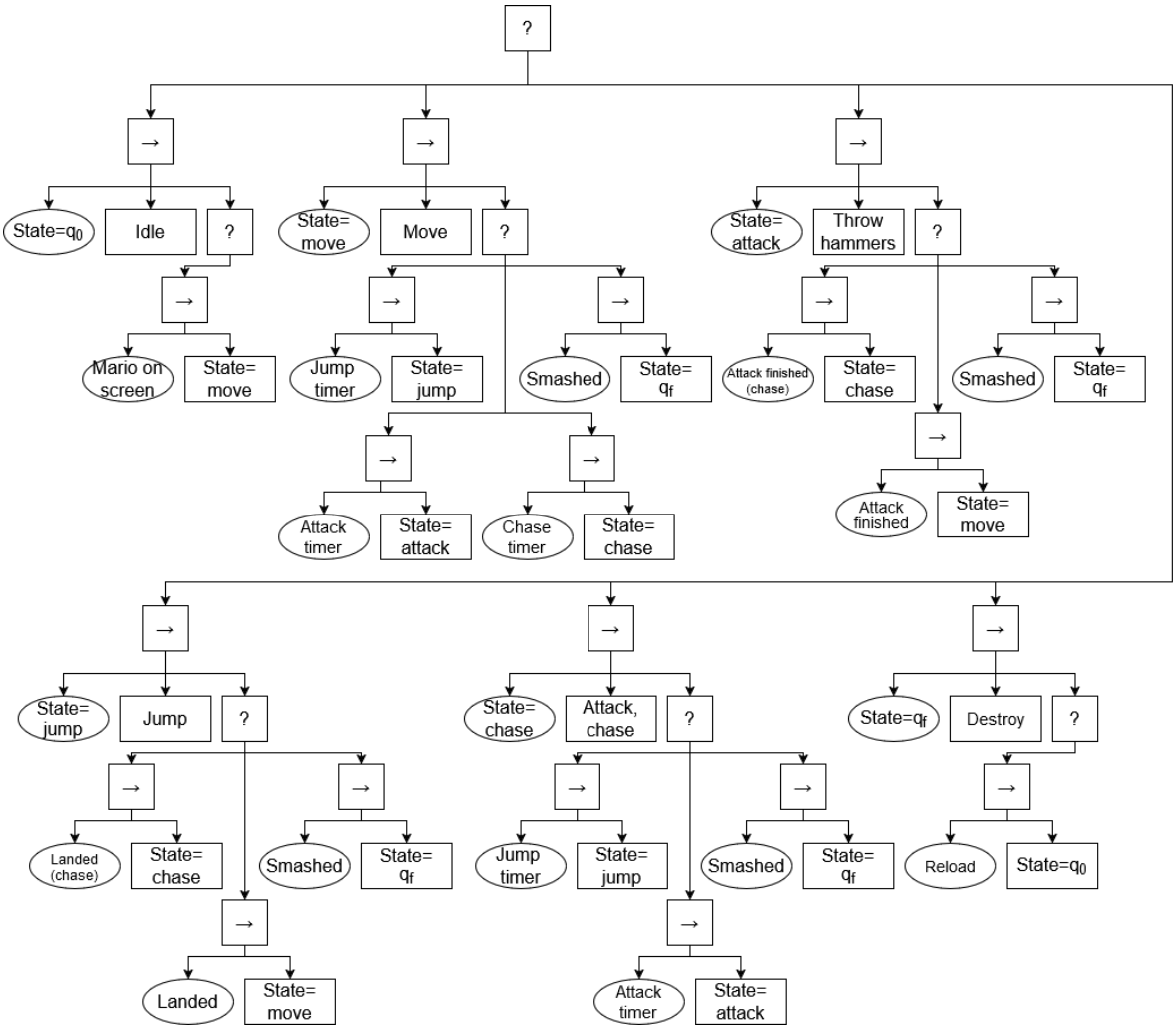

Figure 5.2 : Canonical BT for the Hammer Bro. enemy from *Super Mario Bros.*

5.1 RESULTS AND DISCUSSION

To reiterate, the results from computing the comprehensive difficulty of both the FSM and canonical BT of various enemies from our chosen four games are listed in Table 5.1. First and foremost, as shown for example in Figure 5.5, it is apparent that the scores remain sufficiently proportional across both the FSM and the canonical BT model, validating that these metrics may be compared between both formats through a simple arithmetic conversion.

Figure 5.3 : Original BT output by the ChatGPT prompt for the *Ashina Elite* enemy from *Sekiro: Shadows Die Twice*.

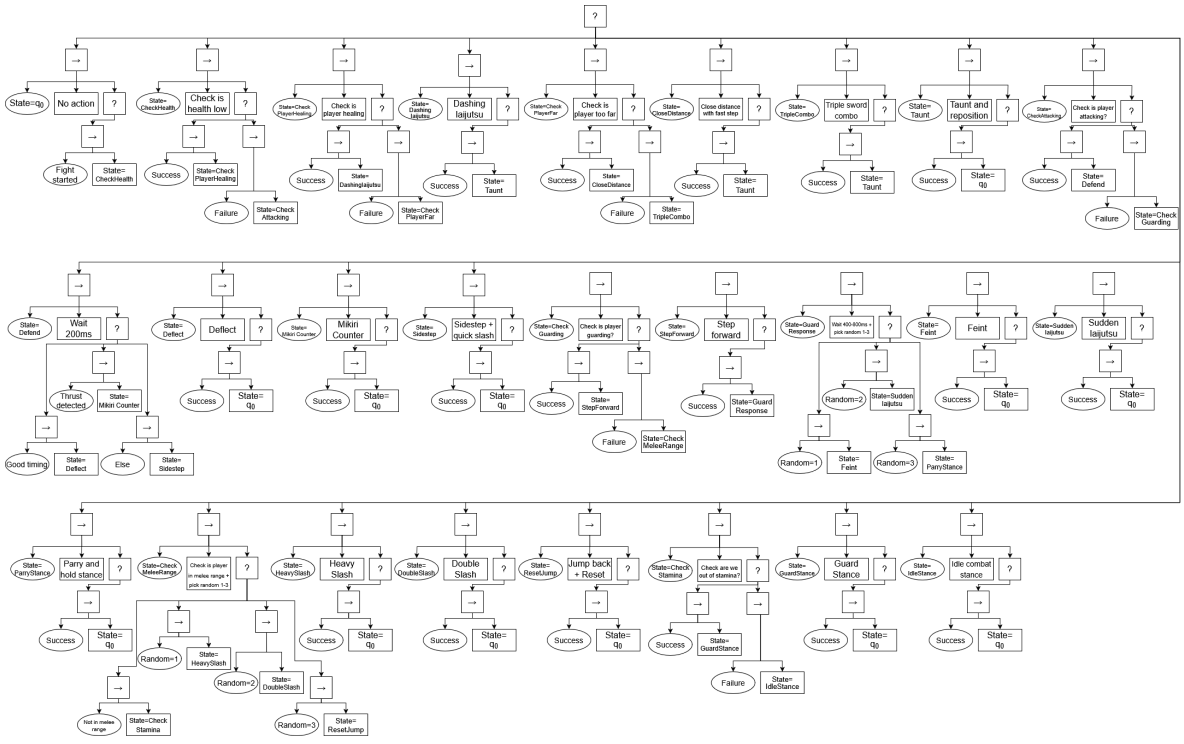

Figure 5.4 : Canonical BT for the *Ashina Elite* enemy from *Sekiro: Shadows Die Twice*.

Table 5.1 : Computed comprehensive difficulty for the FSM and BT of various enemies from *Super Mario Bros.*, *Mega Man*, *Super Punch-Out!!* and *Sekiro: Shadows Die Twice*, respectively

	FSM			BT		
Enemy	n	e	c	n	t	c
Goomba	3	5	9	28	5	140
Koopa	4	8	20	41	8	328
Blooper	5	8	20	45	8	360
Lakitu	5	8	20	45	8	360
Parakoopa	5	10	30	51	10	510
Hammer bro.	6	15	60	70	15	1050
Cut Man	6	8	18	49	8	392
Fire Man	6	7	12	46	7	322
Elec Man	6	8	18	49	8	392
Ice Man	4	4	4	29	4	116
Bomb Man	5	7	15	42	7	294
Guts Man	7	9	21	56	9	504
Bob Charlie	12	16	60	97	16	1552
Dragon Chan	9	12	36	73	12	876
Masked Muscle	9	12	36	73	12	876
Mr Sandman	11	14	44	87	14	1218
Bandit	12	17	72	100	17	1700
Ashina Elite	26	39	364	222	39	8658

Additionally, much of the numbers obtained follow our expectations that enemies fought earlier in their respective games are generally easier to understand: the *Goomba* and the *Koopa*, the two enemies which appear in the first level of *Super Mario Bros.*, both have lower scores than the other four enemies from the same game, which appear later. Both the *Goomba* and the *Koopa* have very simple behaviours which consist of moving forward until they hit an obstacle, with the only difference between the two being that the *Koopa* turns into a shell when stomped rather than dying. Meanwhile, the remaining four enemies have more interesting patterns: the *Blooper* moves up and down depending on its altitude compared to the player's, the *Lakitu* hovers over the player and throws down new enemies, the *Parakoopa* is a flying *Koopa* which loses its wings when stomped the first time, and the *Hammer Bro.*, as mentioned previously, jumps and moves erratically while rapidly throwing hammers at the player, and even chases them if they are too defensive. Likewise, we also suspected that enemies from retro games would indeed be simpler than enemies from modern games, which is corroborated by the enemies from *Sekiro: Shadows Die Twice* scoring much higher than those from the other three.

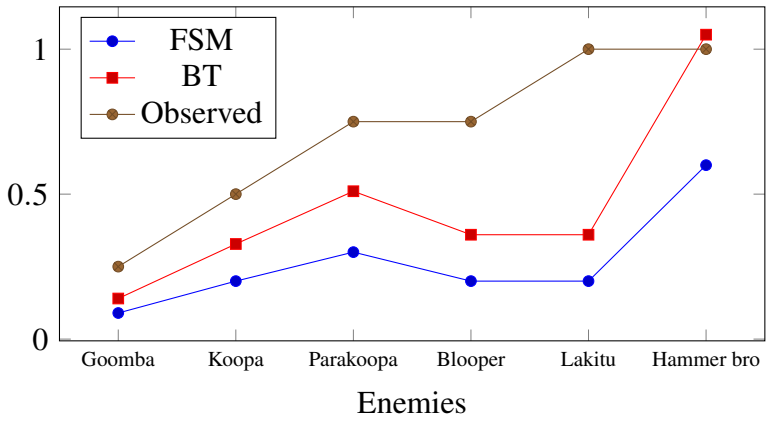

Figure 5.5 : Comprehensive difficulty curve of the *Super Mario Bros.* enemies of Table 5.1

It is of course important to acknowledge that this metric only represents how hard it is for players to grasp how an enemy behaves and not their overall difficulty. This is most obvious when looking at the *Lakitu*, a notoriously difficult enemy who scored in the middle of the pack in our study. This is in large part because of the aforementioned extra enemies it spawns when attacking; while this is relatively simple to understand as a concept, it remains a considerable challenge for players to avoid them and understanding how they work only constitutes a small fraction of the challenge. In a similar manner, it is worth noting that our metric is not confined to scaling in the order the enemies appear in a game; it remains valid whether or not it reflects an enemy's overall difficulty.

In the interest of further validating our findings, we cross-referenced them against public opinions found online through polls on gaming forums and articles. Tables 5.2[67, 68], 5.3[69, 70, 71] and 5.4[72, 73] show the opinions we found; sources include Reddit, GameFAQs, TheTopTens, Steamcommunity and Speedrun.com.

Table 5.2 : Difficulty ranking by players of selected enemies from *Super Mario Bros.* (NES)

Difficulty Tier	**Enemies**
Very Difficult	Hammer Bro, Lakitu
Difficult	Blooper, Parakoopa
Moderate	Koopa Troopa
Easy	Goomba

As suggested earlier, the *Lakitu* is considered a very difficult enemy despite its lower score on our metric. This is because of how it constantly follows the player, dropping enemies which can quickly overwhelm them. In other words, it is a high pressure enemy who requires a higher degree of motor skill to deal with and therefore there is more to its difficulty than just the comprehensive aspect. This is not abnormal; our model specifically targets behavioural complexity. An enemy who is easy to understand is not necessarily easy to handle in terms of game-play, particularly if they are fast or very punishing. See Figures 5.6 and 5.7 for *Lakitu*'s

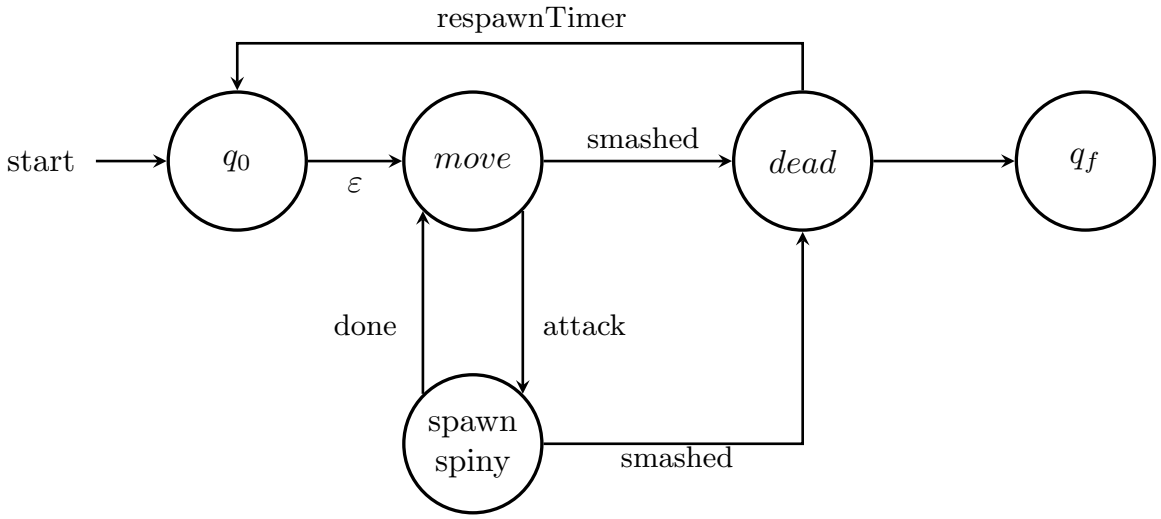

Figure 5.6 : FSM for the Lakitu enemy in *Super Mario Bros.*

FSM and canonical BT. Aside from this, the rest of the rankings largely follows our expected difficulty curve, with the *Hammer Bro.* at the top and the *Koopa* and *Goomba* at the bottom, reinforcing our theory.

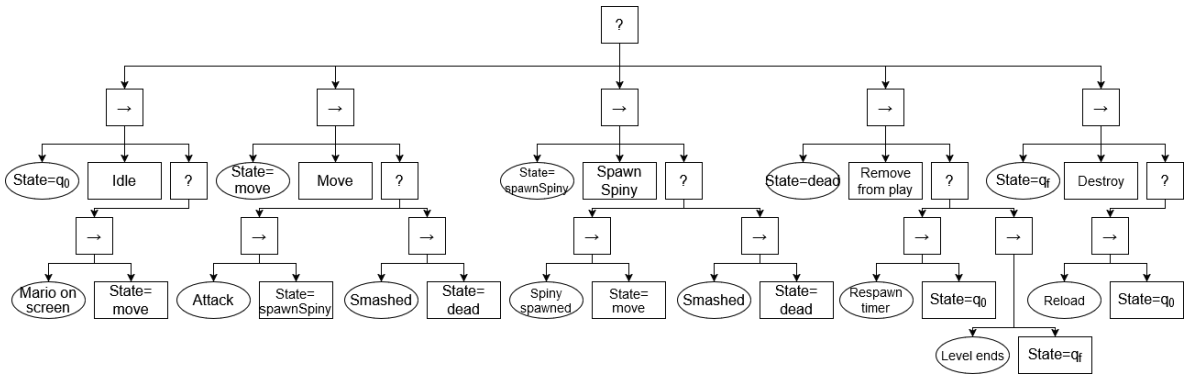

Figure 5.7 : Canonical BT for the Lakitu enemy in *Super Mario Bros.*

Table 5.3 : Difficulty ranking by players of selected boxers from *Super Punch-Out!!* (1994)

Difficulty Tier	Boxers
Very Difficult	Mr. Sandman
Difficult	Masked Muscle
Moderate	Dragon Chan
Easy	Bob Charlie

Table 5.3 ranks *Super Punch-Out!!*'s four Major Circuit boxers from easiest to hardest as Bob Charlie, Dragon Chan, Masked Muscle, and Mr. Sandman. This correlates reasonably with our findings which placed Dragon Chan and Masked Muscle evenly at a score of 876, followed by Mr Sandman at 1218. The only discrepancy is Bob Charlie, who sports a comprehensive difficulty score of 1552. His FSM and his BT may be found in Figures 5.8 and 5.9; recall that the purple transitions represent successes and the orange ones represent failures. There may be a number of reasons for this, but the most likely one, in my eyes, is that he is a very telegraphed fight. All of his punches are preceded by a fairly obvious cue: true to his name Bob Charlie constantly bobs up and down throughout the fight, but stops for a second before striking, giving the player a very helpful heads-up towards dodging his hits. This highlights one of the disconnects between our model and real player perception. Complexity and volume of information are one thing, but whether or not that information is communicated well to the player is another very important aspect which our model does not consider. It is also worth noting that these rankings are sampled from platforms where the user base is already experienced with the game. Such players are unlikely to struggle in these fights and therefore, specific sections of the boxers' FSMs which fire off when the boxer is winning may not come into play. For instance, Bob Charlie has a branch in his behaviour graphs which only occurs when the player is stunned, something which may never happen to an experienced player, potentially skewing the results in favour of fights which can be quickly won through

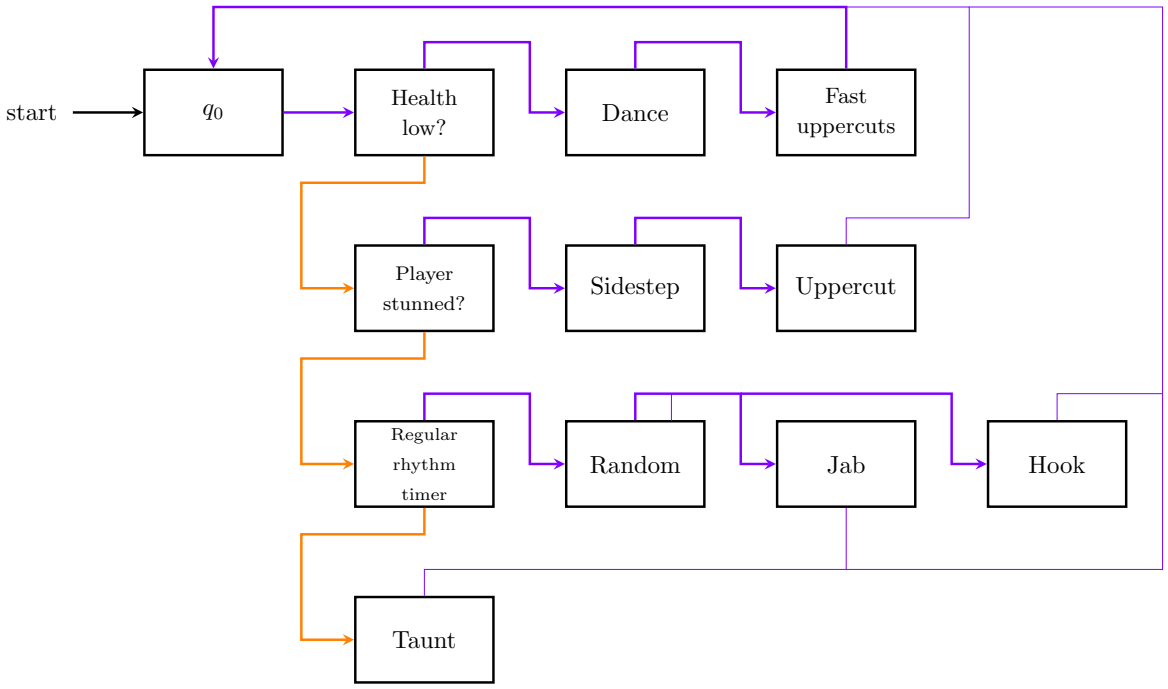

Figure 5.8 : FSM for Bob Charlie from *Super Punch-Out!!*

simple tactics that must be learned beforehand, such as rapidly hitting the enemy in a specific way to keep them stunned.

Mega man is a special case because one of the key features of every game in the series is that the player may choose to fight the first set of bosses in any order. For that reason, we chose to analyze the first set of six bosses from the first game in the series, and our model places them in the following order: Ice man, Bomb Man, Fire Man, Cut Man, Elec Man, and Guts Man. Because the game features a "type weaknesses" system wherein different weapons have different effectiveness against each Robot Master, we stuck to looking at rankings for "buster only" fights, meaning the ranking indicates how difficulty each Robot Master is to defeat with only the starting weapon. Our model correctly places Bomb Man as one of the easier bosses and Elec man as one of the harder ones, but the correlation weakens significantly past that.

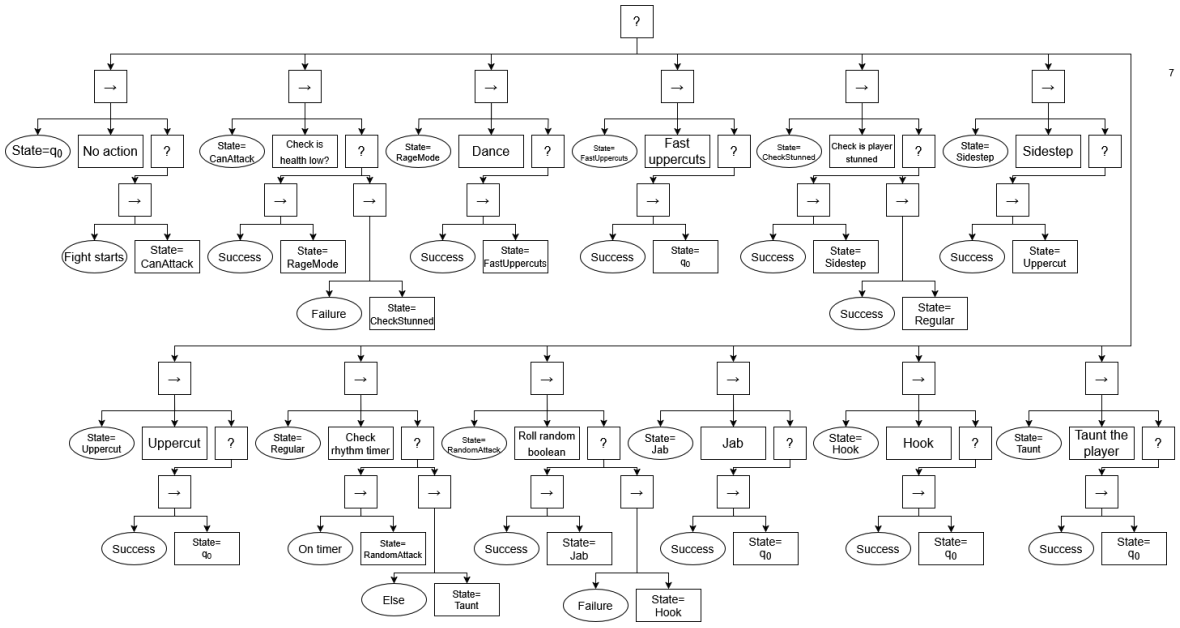

Figure 5.9 : Canonical BT for Bob Charlie from *Super Punch-Out!!*

One particularly egregious discrepancy is Ice Man, who scored very low in our model but is considered a fairly difficult enemy on average, with a significant number of players seeing him as the hardest of the bunch. Like earlier, this could simply be because of a difference in motor difficulty, but I propose another theory: with how low Ice Man scored in the model, this seems to be a good time to highlight the significance of choosing the correct level of granularity for FSMs and BTs. Ice Man's FSM and BT may be found in Figures 5.10 and 5.11. As you can see, one of the nodes simply says "Ice Slasher (x3)", which is a valid way to indicate an action the enemy performs. However, because in reality, when Ice Man performs this action, he floats up or down while firing which causes the projectiles to take on a diagonal formation as they fly towards the player, this is not only significantly harder to dodge for players from a motor skill perspective, but it may also explain why some comments on the ranking's page had players stating they did not perceive a pattern in Ice Man's attacks, despite his attack pattern being quite simple on paper. In other words, there may have been value in splitting the attack

Table 5.4 : Difficulty ranking (ascending order) by players of Mega Man Robot Masters (NES)

Difficulty Order	Robot Masters
1	Cut Man
2	Bomb Man
3	Guts Man
4	Ice Man
5	Fire Man
6	Elec Man

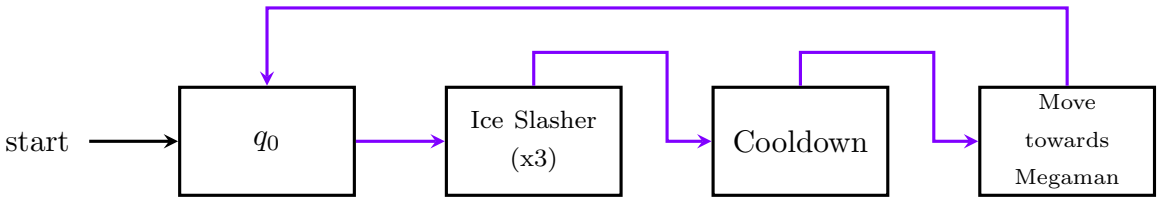

Figure 5.10 : FSM for Ice Man from *Mega Man*

node into multiple, both to indicate that Ice Man floats up or down as he shoots, but also even possibly to separate each attack into its own action, with a self-looping transition.

Lastly, because we only analyzed two enemies from *Sekiro: Shadows Die Twice*, I will not go over community rankings for them, but it is fairly trivial to assess that the very first enemy encountered in the game, the Bandit, is easier than Ashina Elite, a mini-boss found roughly halfway into the game. Nevertheless, interesting insights can be drawn from their results. The Bandit, despite being the first enemy in the game, scored 1700, higher than even the top scoring enemy analyzed in the other three games, and Ashina Elite scored 8658, eclipsing the rest entirely. This is partially because on top of being the only 3D game analyzed, games such as *Sekiro: Shadows Die Twice*, known as souls-like games, are notoriously difficult. However, this score difference can also be explained by how modern games are generally

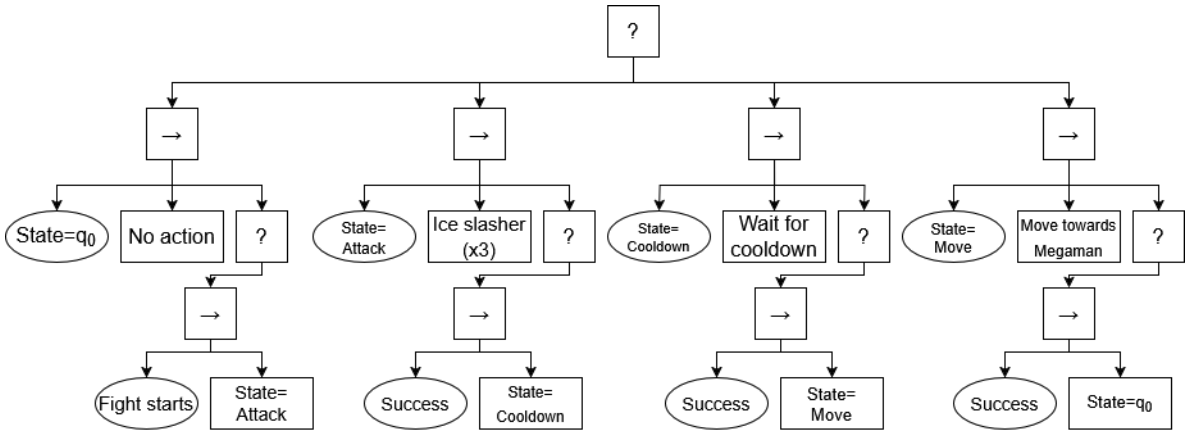

Figure 5.11 : Canonical BT for Ice Man from *Mega Man*

more complex across the board. Indeed, if we were to map the average complexity of enemies in each game, we could see that both games released in the 80's, *Super Mario Bros.* and *Mega Man*, scored the lowest on average, while *Super Punch-Out!!*, released a decade later, features an average score of roughly double that of the previous two, and finally *Sekiro: Shadows Die Twice* ranks well above all three.

CONCLUSION

Modern day video games have become a very mainstream hobby, as well as an equally large industry. However, as the industry grows, so do both the complexity and the standard of quality of its products, and with them, the production time and costs also increase. The testing phase of a project is no exception to this and unfortunately, to this day the testing process still largely consists of slow and repetitive manual labour as play-testers must assess whether the game is fun and balanced. My master's thesis proposes a solution to this by putting together a model for the automated assessment of video game difficulty. Video game difficulty can be divided into multiple different subtypes, with the three main types being motor, strategic and comprehensive difficulty. The solution proposed within my master's thesis focuses on the latter, specifically that of the enemies encountered by the player in the video game, by exploiting properties of the already popular graph models used to represent non-player character behaviours: the Finite State Machine (FSM) and the Behaviour Tree (BT). As with all graphs, these two models bear a metric known as cyclomatic complexity, or cycle rank. Through an equation combining a behaviour graph's cyclomatic complexity and its number of nodes, we can obtain a formal and reproducible metric which can be used to quantify the intrinsic difficulty in understanding an enemy's behaviour.

My master's thesis then provides a set of algorithms which I've transcribed to pseudo-code from Colledanchise and Ögren's book on BTs[39]. These algorithms enable the conversion of FSMs and BTs into one another, and I prove their adequacy in producing results whose comprehensive difficulty metric consistently falls within the same function order as their source graph, which opens the possibility for direct comparison of the metric between the two models. Next, I cover a case study we conducted in which we used our model to analyze a set of enemies from four well-known games: *Super Mario Bros.*, *Mega Man*, *Super Punch-Out!!*,

and *Sekiro: Shadows Die Twice*. The model showed promising accuracy in measuring the enemies' comprehensive difficulty as it matched expected difficulty curves and a satisfactory number of community rankings. The model and the algorithms were also featured within both an IEEE conference as well as an article in the Elsevier Computing journal.

The model still harbours some limitations however, as it fails to account for various aspects of video games as a whole. Particularly, it ignores the other two main dimensions in the difficulty space which are motor and strategic difficulty. Furthermore, while it aims to disregard player-centric difficulty in order to keep it objective, it does also abstract difficulty which stems from player-enemy interactions, such as how the visuals of an enemy's ability influence whether the player successfully understands what it does.

Future work could certainly contribute to improving the model's coverage in that direction. Other avenues include the possibility of integrating the model into popular game engines as an add-on, which should be relatively free of any interface friction seeing as BTs and FSMs are already the norm for NPC programming within most engines. It would also be worthwhile to further explore how the model handles randomness and how it affects the comprehensive difficulty of an entity, as it is a common tool in video games. Another interesting lead would be to adapt the model to the more complex modern AI systems which make use of machine learning. I also believe it would make sense to examine other metrics or even models, as I encountered a few during my literature review which I did not have time to explore in depth. For instance, I believe Cognitive Load Theory could teach us concepts which could be helpful in this projects. I also encountered a model known as the Petri Net, which I feel could prove useful in modelling player-enemy interactions. Last but not least, we should of course not underestimate the importance of continuing to perform validation analyses with further datasets which include genres not visited in this master's thesis, as pinpointing exactly where the model falters will be crucial to reaching a complete understanding of the issue.

REFERENCES

[1] S. Zoting, "Video games market size to hit usd 721.77 billion by 2034," Precedence Research Pvt. Ltd., Tech. Rep., Jan 31 2025. [Online]. Available: https://www.precedenceresearch.com/video-game-market

[2] Z. Kachwala, "Newzoo tempers global game market forecast amid sluggish console sales," *Reuters*, 2024. [Online]. Available: https://www.reuters.com/technology/newzoo-tempers-global-game-market-forecast-amid-sluggish-console-sales-2024-08-13/?utm_source=chatgpt.com

[3] R. J. Haier, B. V. Siegel Jr, A. MacLachlan, E. Soderling, S. Lottenberg, and M. S. Buchsbaum, "Regional glucose metabolic changes after learning a complex visuospatial/motor task: a positron emission tomographic study," *Brain research*, vol. 570, no. 1-2, pp. 134–143, 1992.

[4] C. S. Green and D. Bavelier, "Action video game modifies visual selective attention," *Nature*, vol. 423, no. 6939, pp. 534–537, 2003.

[5] C. Basak, W. R. Boot, M. W. Voss, and A. F. Kramer, "Can training in a real-time strategy video game attenuate cognitive decline in older adults?" *Psychology and aging*, vol. 23, no. 4, p. 765, 2008.

[6] J. A. Anguera, J. Boccanfuso, J. L. Rintoul, O. Al-Hashimi, F. Faraji, J. Janowich, E. Kong, Y. Larraburo, C. Rolle, and E. Johnston, "Video game training enhances cognitive control in older adults," *Nature*, vol. 501, no. 7465, pp. 97–101, 2013.

[7] M. Özçetin, F. Gümüstas, Y. Çag, I. Z. Gökbay, and A. Özmel, "The relationships between video game experience and cognitive abilities in adolescents," *Neuropsychiatric disease and treatment*, pp. 1171–1180, 2019.

[8] E. Stanmore, B. Stubbs, D. Vancampfort, E. D. de Bruin, and J. Firth, "The effect of active video games on cognitive functioning in clinical and non-clinical populations: A meta-analysis of randomized controlled trials," *Neuroscience & Biobehavioral Reviews*, vol. 78, pp. 34–43, 2017.

[9] A. Fishman, "Video games are social spaces: how video games help people connect,"

Psychology today, vol. 22, 2019. [Online]. Available: https://www.psychologytoday.com/us/blog/video-game-health/201901/video-games-are-social-spaces

[10] A. Nicholson, "Video games and social connection in the digital age," *VGKAMI*, 2023. [Online]. Available: https://vgkami.com/video-games-and-social-connection-in-the-digital-age/

[11] B. K. Wiederhold, "Kids will find a way: The benefits of social video games," pp. 213–214, 2021.

[12] M. Csikszentmihalyi, *Flow*. HarperCollins, 1991. [Online]. Available: https://books.google.ca/books?id=v2AVz3gf-F4C

[13] R. Frampton, "A new taxonomy of difficulty: A call to reconsider how we talk about difficulty in games." *GameDeveloper.com*, pp. 1–7, 2021.

[14] A. Denisova, C. Guckelsberger, and D. Zendle, "Challenge in digital games: Towards developing a measurement tool," in *Proceedings of the 2017 chi conference extended abstracts on human factors in computing systems*, 2017, pp. 2511–2519.

[15] D. Dicere, "Artificial difficulty in games," 2024. [Online]. Available: https://indiecator.org/2024/08/20/artificial-difficulty-in-games/

[16] B. Wirtz, "Artificial difficulty in games: What is it?" *Game Designing*, 2023. [Online]. Available: https://www.gamedesigning.org/learn/artificial-difficulty/

[17] M. Preece, "The difference between challenge and artificial difficulty," no. June 2, 2025, 2022. [Online]. Available: https://mygaming.co.za/news/pc/125192-the-difference-between-challenge-and-artificial-difficulty

[18] K. Ballard, "Let's play ftl: Faster than light part 3-4 (patreon chosen game)," 2016. [Online]. Available: https://youtu.be/2bnuMPMl6SE?list=PL5dr1EHvfwpMw4EWheHqih_fk_1YTywI1&t=1930

[19] N. Fisher and A. K. Kulshreshth, "Exploring dynamic difficulty adjustment methods for

video games," in *Virtual Worlds*, vol. 3. MDPI, 2024, pp. 230–255.

[20] D. Dziedzic and W. Włodarczyk, "Approaches to measuring the difficulty of games in dynamic difficulty adjustment systems," *International Journal of Human–Computer Interaction*, vol. 34, no. 8, pp. 707–715, 2018.

[21] W. Cai, M. Chen, and V. C. Leung, "Toward gaming as a service," *IEEE Internet Computing*, vol. 18, no. 3, pp. 12–18, 2014.

[22] P. Promsutipong and V. Kotrajaras, "Enemy evaluation ai for 2d action-platform game," in *2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. IEEE, 2017, pp. 1–6.

[23] J. Zhang, "Directly controlling the perceived difficulty of a shooting game by the addition of fake enemy bullets," in *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–5.

[24] Y. Francillette, H. Tremblay, and B. Bouchard, "Automated difficulty assessment model for comprehensive difficulty in games," in *2024 IEEE Gaming, Entertainment, and Media Conference (GEM)*, 5-7 June 2024 2024, pp. 1–6.

[25] G. Tavinor, "The definition of videogames revisited," in *The Philosophy of Computer Games Conference, Oslo*, 2009, pp. 1–11.

[26] ——, "Definition of videogames," *Contemporary Aesthetics (Journal Archive)*, vol. 6, no. 1, p. 16, 2008.

[27] J. Velikovsky, "Flow theory, evolution & creativity: or,'fun & games'," in *Proceedings of the 2014 Conference on interactive entertainment*, 2014, pp. 1–10.

[28] M. Csikszentmihalyi, *Beyond boredom and anxiety*. Jossey-bass, 2000.

[29] M. Csikszentmihalyi and F. Massimini, "On the psychological selection of bio-cultural information," *New Ideas in Psychology*, 1985.

[30] M. Csikszentmihalyi and M. Csikzentmihaly, *Flow: The psychology of optimal experience*. Harper & Row New York, 1990, vol. 1990.

[31] M. Csikszentmihalyi, "Flow and the psychology of discovery and invention," *HarperPerennial, New York*, vol. 39, pp. 1–16, 1997.

[32] G. Polančič and B. Cegnar, "Complexity metrics for process models - a systematic literature review," *Computer Standards & Interfaces*, vol. 51, pp. 104–117, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.csi.2016.12.003

[33] J. L. Gross, J. Yellen, and M. Anderson, *Graph theory and its applications*. Chapman and Hall/CRC, 2005.

[34] T. J. McCabe, "A complexity measure," *IEEE Trans. Softw. Eng.*, vol. 2, no. 4, p. 308–320, Jul. 1976.

[35] M. Nicolau, D. Perez-Liebana, M. O'Neill, and A. Brabazon, "Evolutionary behavior tree approaches for navigating platform games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 3, pp. 227–238, 2016.

[36] J. A. Bagnell, F. Cavalcanti, L. Cui, T. Galluzzo, M. Hebert, M. Kazemi, M. Klingensmith, J. Libby, T. Y. Liu, and N. Pollard, "An integrated system for autonomous robotics manipulation," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 2955–2962.

[37] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, and G. D. Hager, "Costar: Instructing collaborative robots with behavior trees and vision," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 564–571.

[38] J. K. Gershenson, G. Prasad, and Y. Zhang, "Product modularity: definitions and benefits," *Journal of Engineering design*, vol. 14, no. 3, pp. 295–313, 2003.

[39] M. Colledanchise and P. Ögren, *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

[40] C. Simpson, "Behavior trees for ai: How they work," *Game Developer*, 2014. [Online]. Available: https://www.gamedeveloper.com/programming/behavior-trees-for-ai-how-they-work

[41] B. Hannaford, D. Hu, D. Zhang, and Y. Li, "Simulation results on selector adaptation in behavior trees," *arXiv preprint arXiv:1606.09219*, 2016.

[42] A. Denisova, C. Guckelsberger, and D. Zendle, "Challenge in digital games: Towards developing a measurement tool," in *Proceedings of the 2017 chi conference extended abstracts on human factors in computing systems*, 2017, pp. 2511–2519.

[43] M.-V. Aponte, G. Levieux, and S. Natkin, "Measuring the level of difficulty in single player video games," *Entertainment Computing*, vol. 2, no. 4, pp. 205–213, 2011.

[44] J. T. Kristensen, A. Valdivia, and P. Burelli, "Statistical modelling of level difficulty in puzzle games," in *2021 IEEE Conference on Games (CoG)*. IEEE Press, 2021, p. 1–8.

[45] M.-V. Aponte, G. Levieux, and S. Natkin, "Difficulty in videogames: an experimental validation of a formal definition," in *Proceedings of the 8th international conference on advances in computer entertainment technology*, 2011, pp. 1–8.

[46] M.-V. Aponte, "Difficulty in videogames: An experimental validation of a formal definition," in *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*, ser. ACE '11. New York, NY, USA: Association for Computing Machinery, 2011.

[47] P. D. Paraschos and D. E. Koulouriotis, "Game difficulty adaptation and experience personalization: A literature review," *International Journal of Human–Computer Interaction*, vol. 39, no. 1, pp. 1–22, 2023.

[48] P. Kaimara, "Development of a scale for measuring the learning experience in serious games," *Digital Culture & Audiovisual Challenges, Interdisciplinary Creativity In Arts and Technology*, pp. 1–7, 2019.

[49] J. L. Aurentz, A. M. Navarro, and D. R. Insua, "Learning the rules of the game: An interpretable ai for learning how to play," *IEEE Transactions on Games*, vol. 14, no. 2, pp.

253–261, 2022.

[50] N. Fisher and A. K. Kulshreshth, "Exploring dynamic difficulty adjustment methods for video games," *Virtual Worlds*, vol. 3, no. 2, pp. 230–255, 2024.

[51] S. Iftikhar, M. Z. Iqbal, M. U. Khan, and W. Mahmood, "An automated model based testing approach for platform games," in *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2015, pp. 426–435.

[52] Y. F. H. T. B. B. S. L. M. R. J. Linard, "Automated difficulty assessment model for platformer games: A comprehensive approach," in *2023 IEEE CTSoc Gaming, Entertainment and Media conf.*, 2023, pp. 1–6.

[53] M. Ouellette, L. Breeding, and C. Clark, "Using applied cognitive load theory and difficulty analysis for educational game design for understanding and transference of literacy skills in adults," in *Proc. of the 14th Int. Conf. on the Foundations of Digital Games*, ser. FDG '19. New York, NY, USA: Association for Computing Machinery, 2019.

[54] Y. J. Kim and J. A. Ruipérez-Valiente, "Data-driven game design: The case of difficulty in educational games." Berlin, Heidelberg: Springer-Verlag, 2020, p. 449–454.

[55] Y. Francillette, H. Tremblay, and B. Bouchard, "Automated difficulty assessment model for comprehensive difficulty in games," in *2024 IEEE Gaming, Entertainment, and Media Conference (GEM)*, 2024, pp. 1–6.

[56] A. B. Jaffe, "Understanding game balance with quantitative methods," Ph.D. dissertation, University of Washington, 2013.

[57] V. Hom and J. Marks, "Automatic design of balanced board games," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 3, 2007, pp. 25–30.

[58] P. Promsutipong and V. Kotrajaras, "Enemy evaluation ai for 2d action-platform game," in *2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2017, pp. 1–6.

[59] CASAS, “Reading basic skills content standards by instructional level,” 2009. [Online]. Available: https://www.casas.org/docs/default-source/research/reading-content-standards-(2009).pdf?sfvrsn=7?Status=Master

[60] A. J. A. Seyderhelm and K. L. Blackmore, “How hard is it really? assessing game-task difficulty through real-time measures of performance and cognitive load,” *Simulation & Gaming*, vol. 54, no. 3, pp. 294–321, 2023. [Online]. Available: https://sbiproxy.uqac.ca/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=163684143&lang=fr&site=ehost-live

[61] I. O. f. Standardization, “Road vehicles—transport information and control systems—detection-response task (drt) for assessing attentional effects of cognitive load in driving,” 2016.

[62] A. Vandierendonck, “A comparison of methods to combine speed and accuracy measures of performance: A rejoinder on the binning procedure,” *Behavior research methods*, vol. 49, no. 2, pp. 653–673, 2017.

[63] J. Zhang, “Directly controlling the perceived difficulty of a shooting game by the addition of fake enemy bullets,” in *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, ser. CHI EA ’21. New York, NY, USA: Association for Computing Machinery, 2021.

[64] B. M. F. Viana, L. T. Pereira, and C. F. M. Toledo, “Illuminating the space of enemies through map-elites,” in *2022 IEEE Conference on Games (CoG)*. IEEE Press, 2022, p. 17–24. [Online]. Available: https://doi.org/10.1109/CoG51982.2022.9893621

[65] T. Anne and J.-B. Mouret, “Parametric-task map-elites,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO ’24. New York, NY, USA: Association for Computing Machinery, 2024, p. 68–77. [Online]. Available: https://doi.org/10.1145/3638529.3653993

[66] L. Spierewka, R. Szrajber, and D. Szajerman, “Procedural level generation with difficulty level estimation for puzzle games,” in *Computational Science – ICCS 2021*, M. Paszynski, D. Kranzlmüller, V. V. Krzhizhanovskaya, J. J. Dongarra, and P. M. A. Sloot, Eds. Springer International Publishing, 2021// 2021, pp. 106–119.

[67] T. T. Tens, “Hardest mario enemies ranked,” https://www.thetoptens.com/super-mario/hardest-mario-enemies/, 2023, accessed June 2025.

[68] R. community, “Who is the most annoying enemy in mario?” https://www.reddit.com/r/Mario/comments/16uw9zs/most_annoying_mario_enemy_in_your_opinion/, 2023, accessed June 2025.

[69] Speedrun.com, “Super punch-out!! individual boxer times,” https://www.speedrun.com/spo, 2024, accessed June 2025.

[70] G. Users, “Super punch-out!! - who’s the hardest opponent?” https://gamefaqs.gamespot.com/boards/954363-punch-out/49663752, 2009, accessed June 2025.

[71] R. community, “Super punch-out!! tier list discussion,” https://www.reddit.com/r/punchout/comments/1acu8ho/super_punch_put_difficulty_tier_list/, 2024, accessed June 2025.

[72] Reddit, “Poll: Who’s the hardest robot master in all of mega man?” https://www.reddit.com/r/Megaman/comments/1ec8amo/mega_man_1_robot_master_busteronly_difficulty/, 2023, accessed June 2025.

[73] S. INC, “Robot master difficulty ranking discussion,” https://sprites-inc.co.uk/thread-329-post-12909.html, 2023, accessed June 2025.

APPENDIX A

ALL FSMS AND BTS GENERATED BY CONVERSION ALGORITHMS

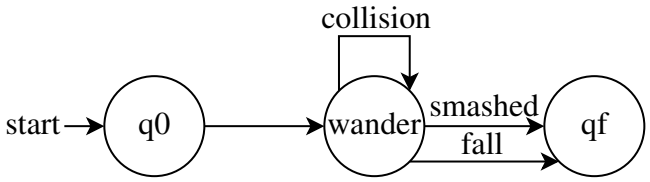

Figure A.1 : Goomba FSM

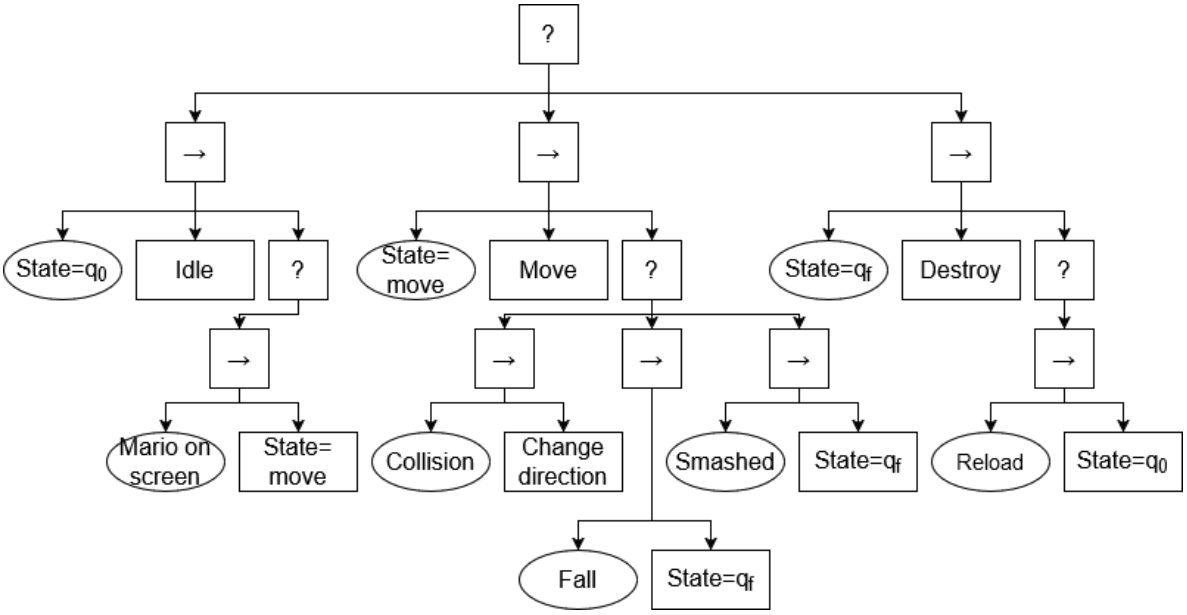

Figure A.2 : Goomba BT

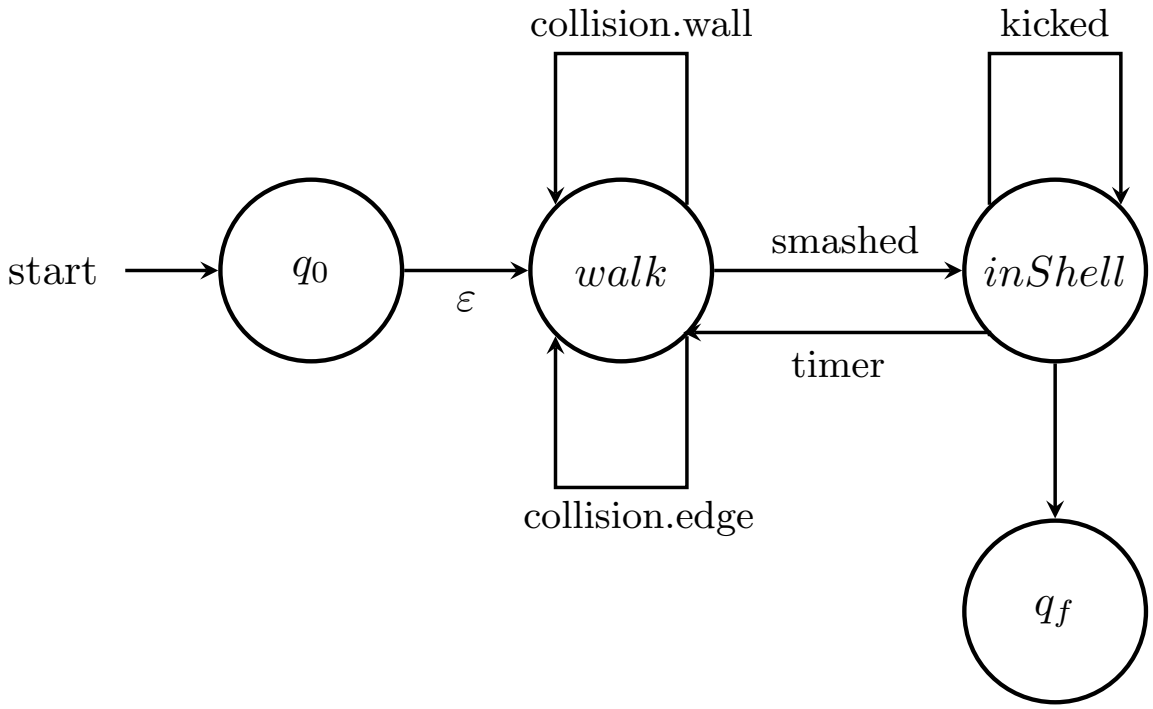

Figure A.3 : Koopa FSM

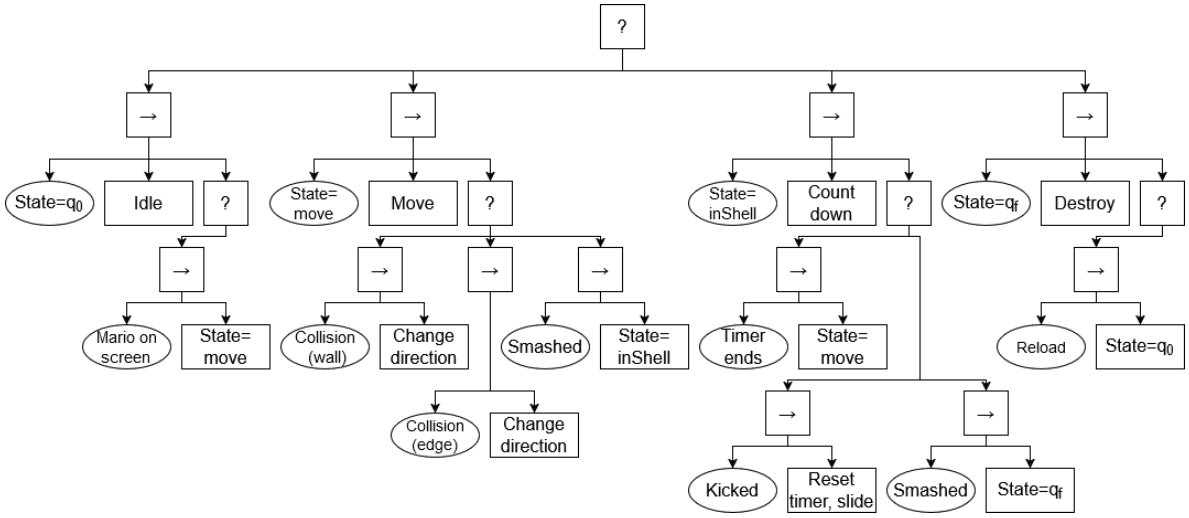

Figure A.4 : Koopa BT

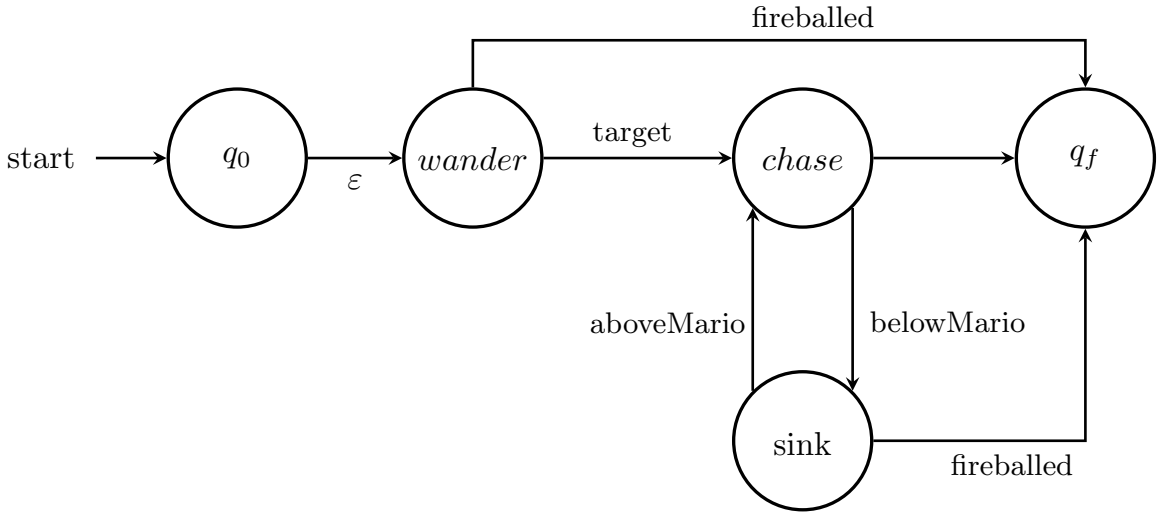

Figure A.5 : Blooper FSM

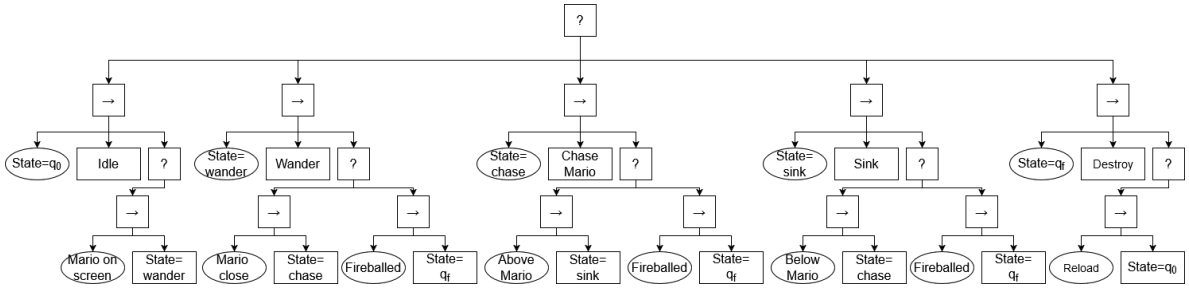

Figure A.6 : Blooper BT

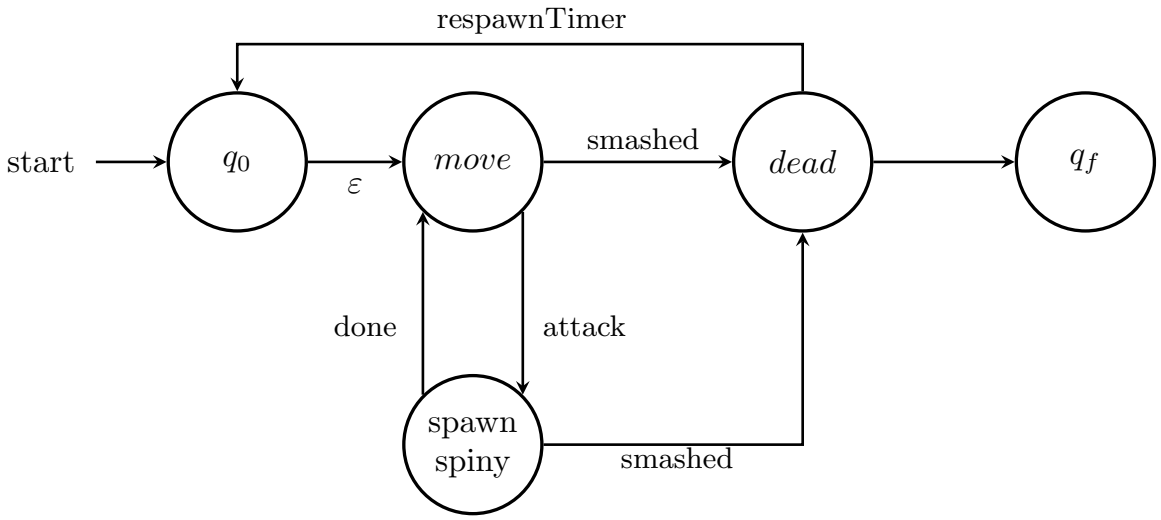

Figure A.7 : Lakitu FSM

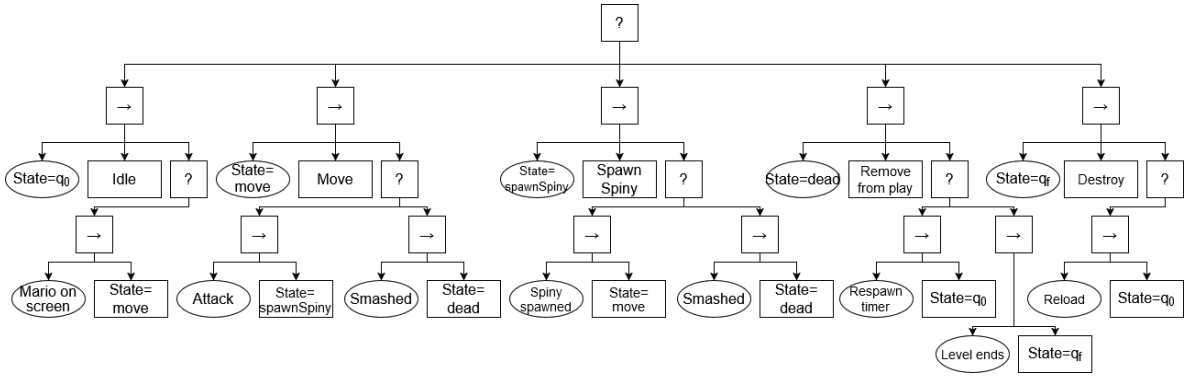

Figure A.8 : Lakitu BT

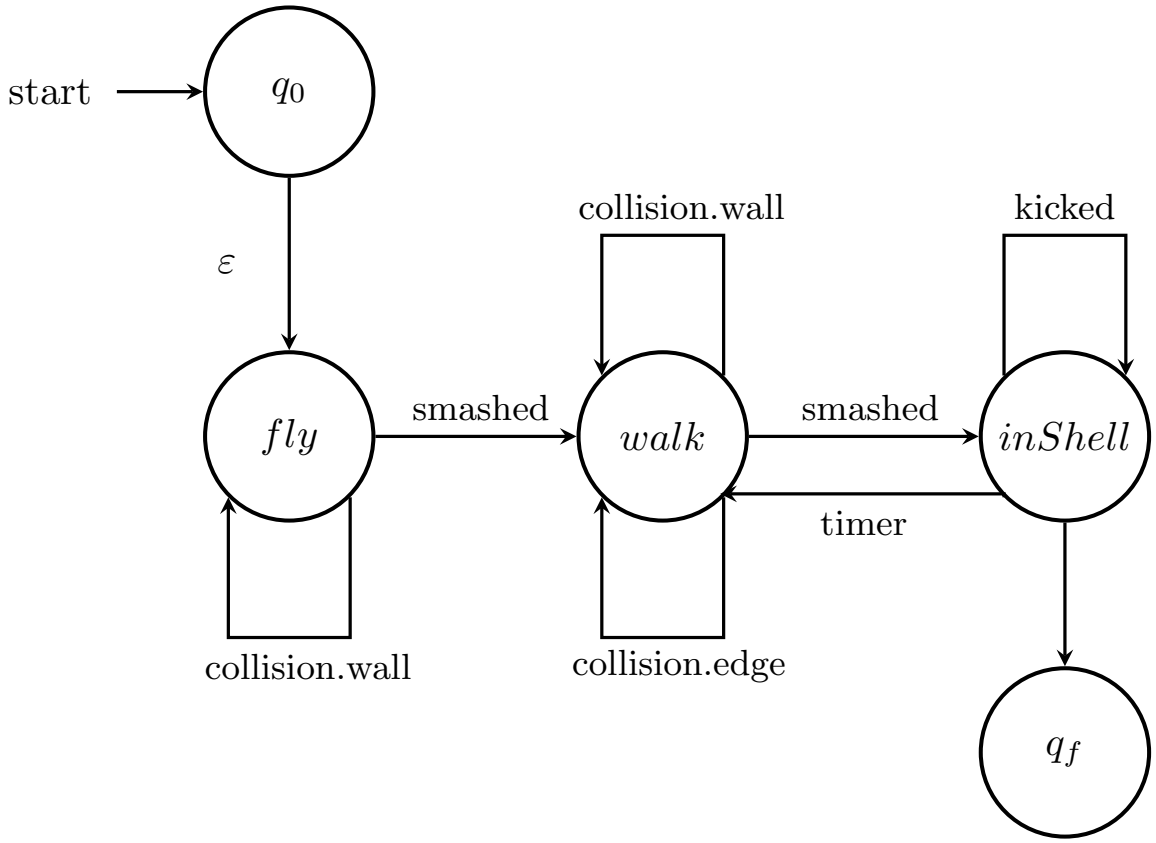

Figure A.9 : Parakoopa FSM

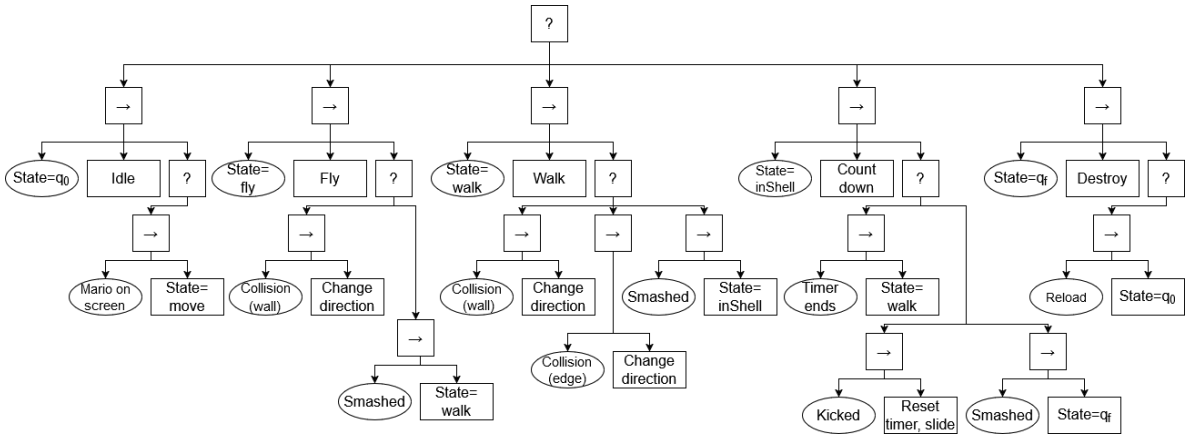

Figure A.10 : Parakoopa BT

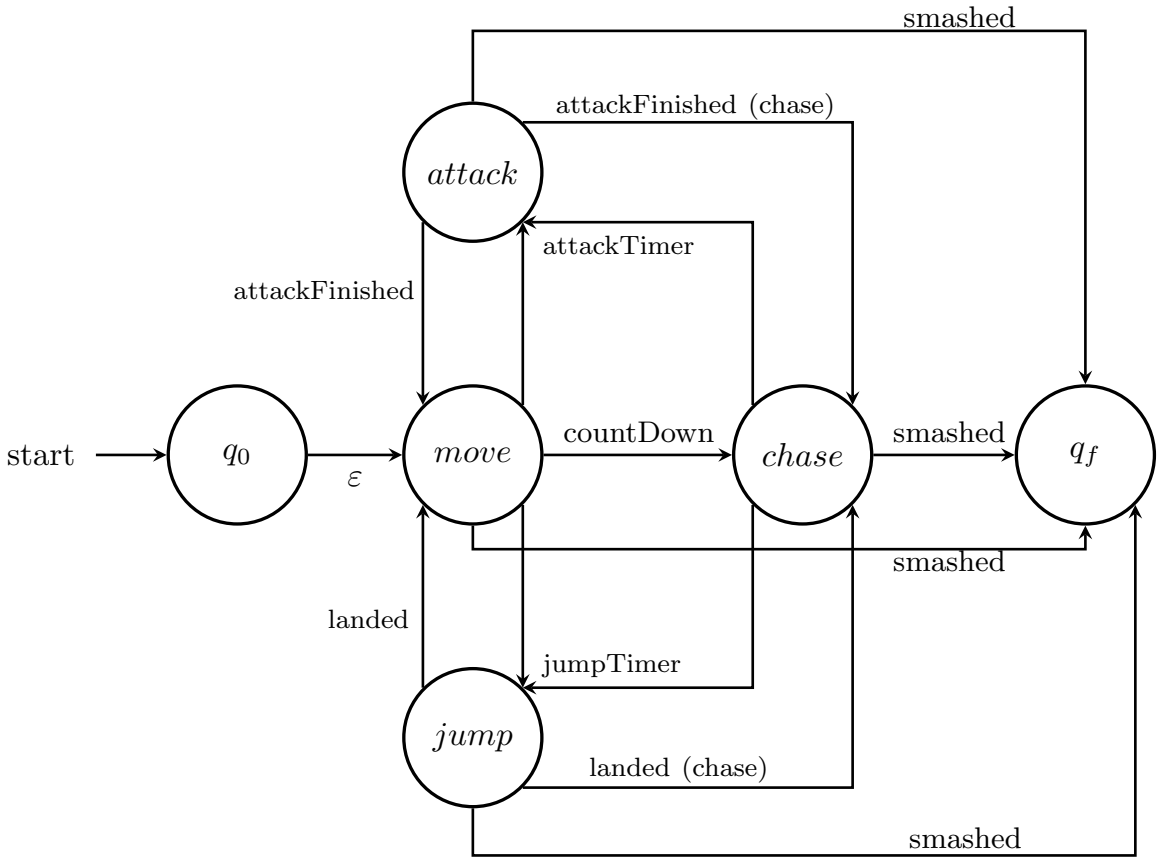

Figure A.11 : Hammer Bro FSM

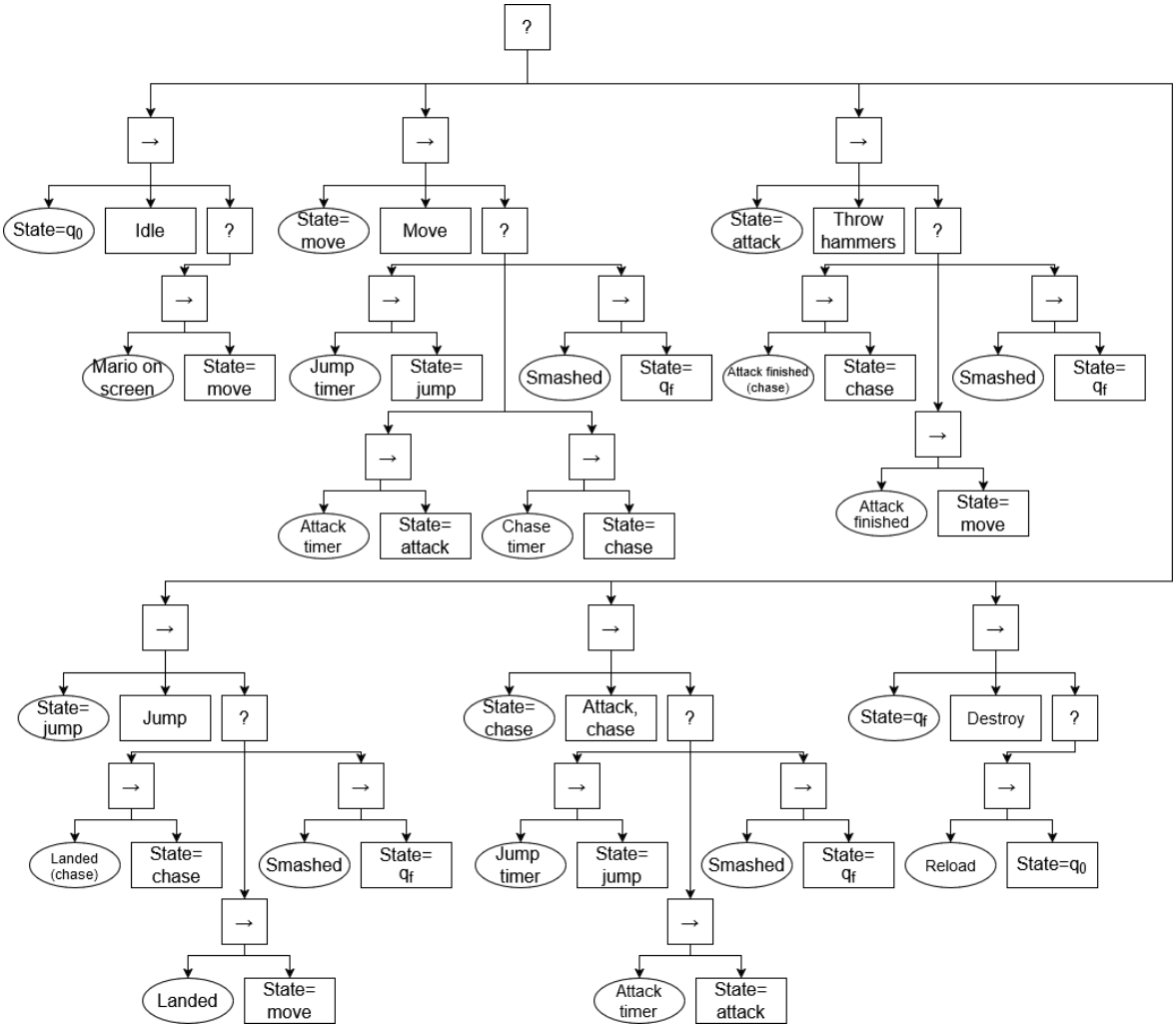

Figure A.12 : Hammer Bro BT

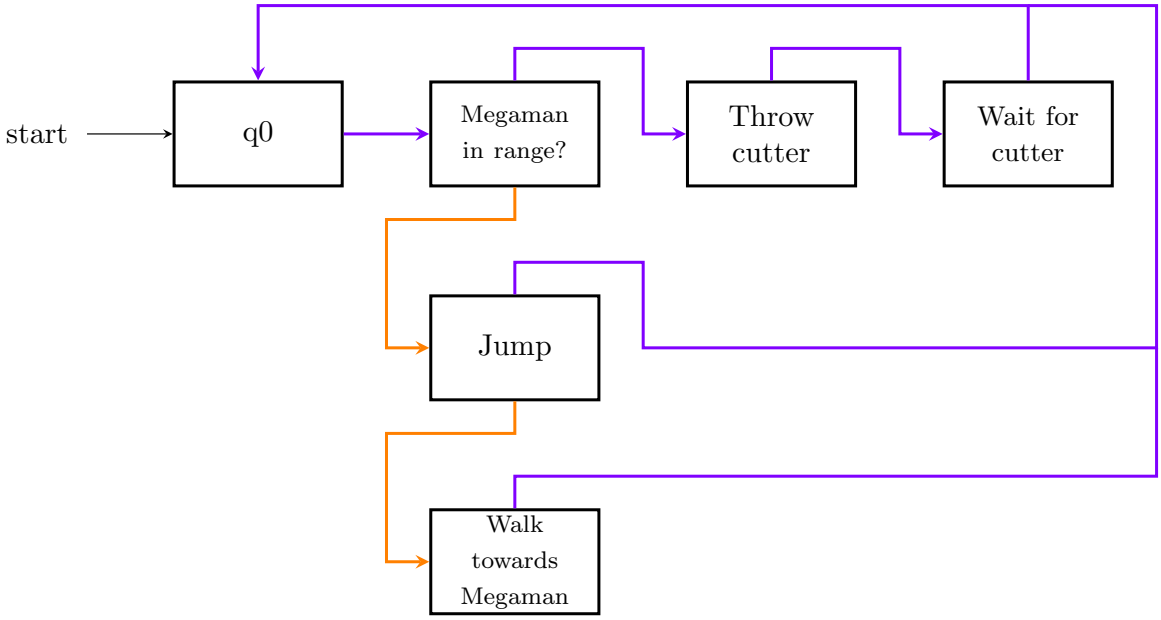

Figure A.13 : Cut Man FSM

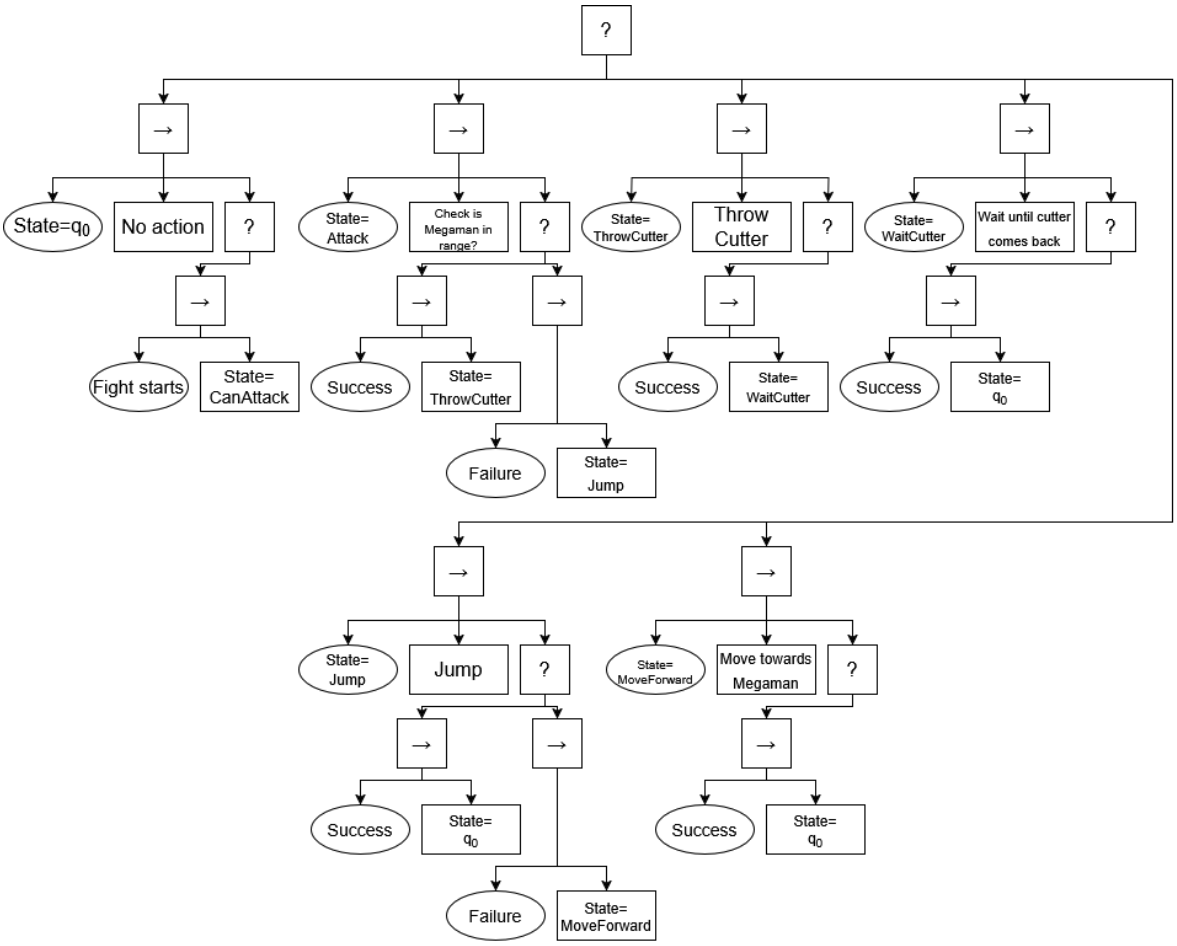

Figure A.14 : Cut Man BT

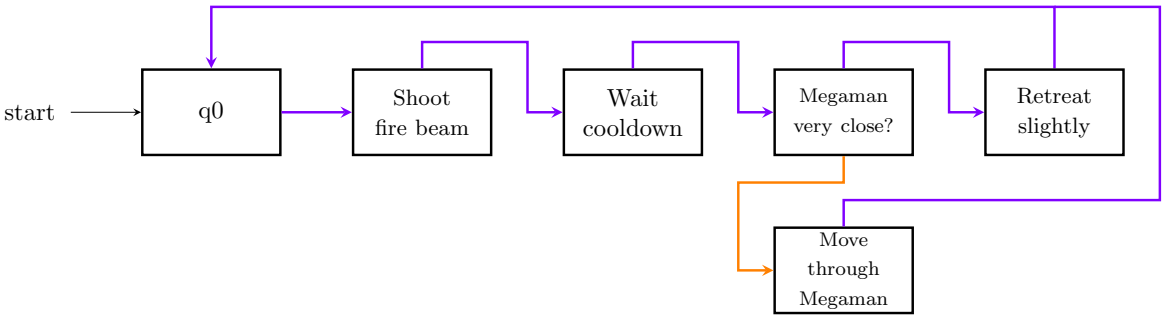

Figure A.15 : Fire Man FSM

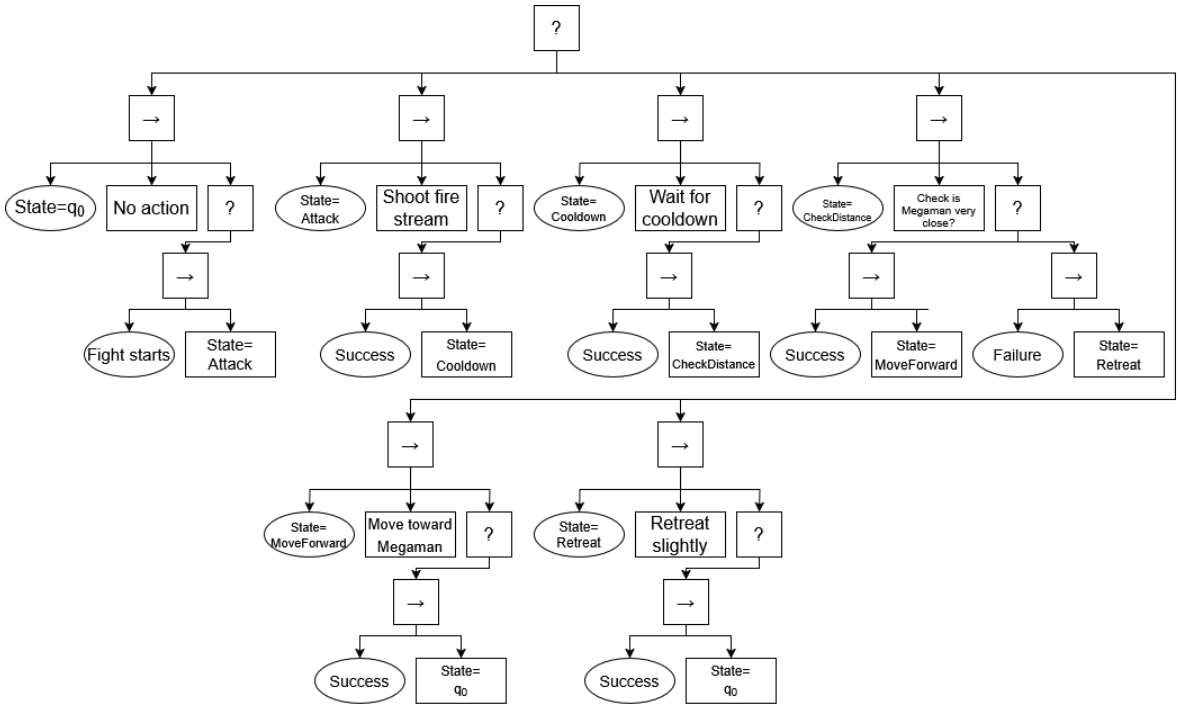

Figure A.16 : Fire Man BT

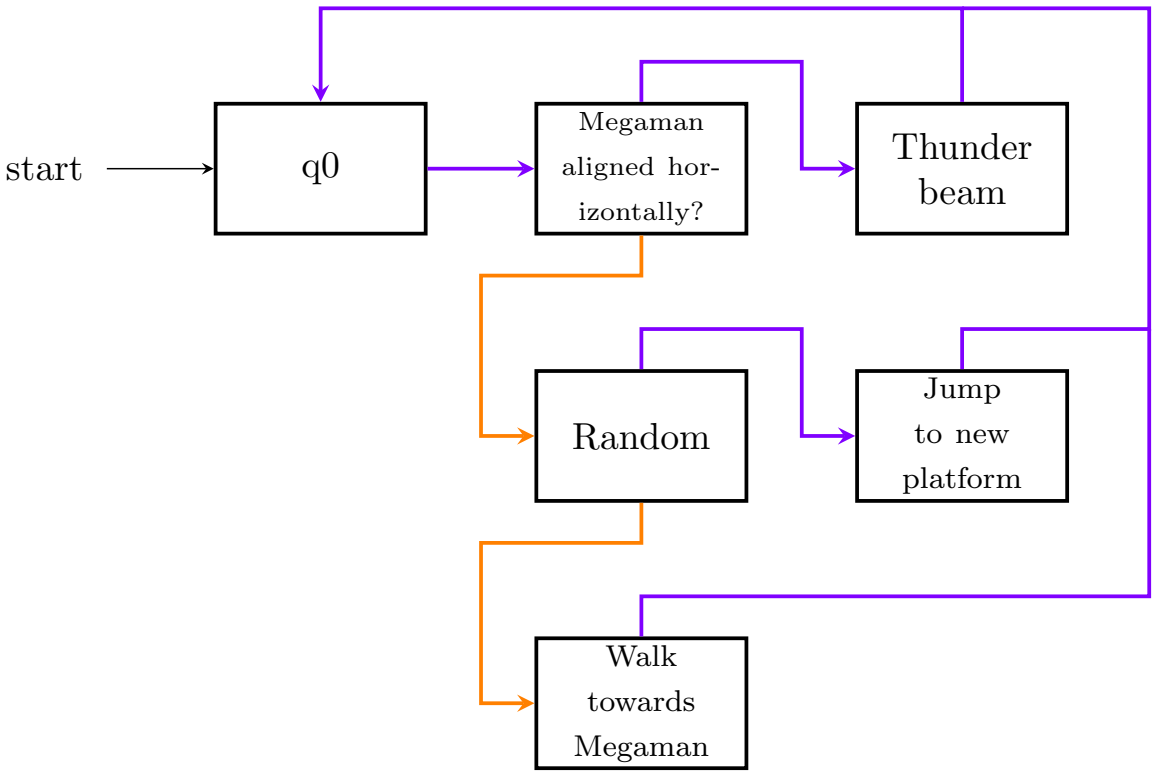

Figure A.17 : Elec Man FSM

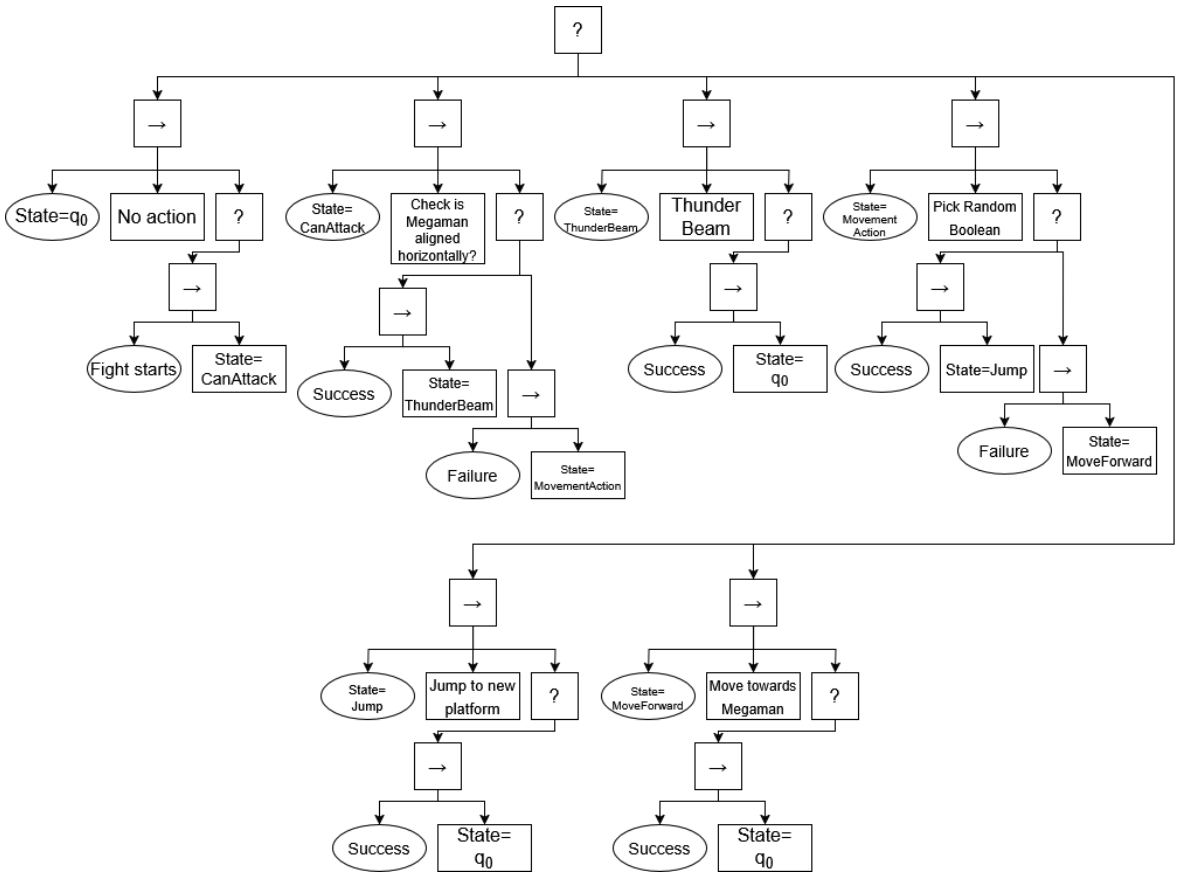

Figure A.18 : Elec Man BT

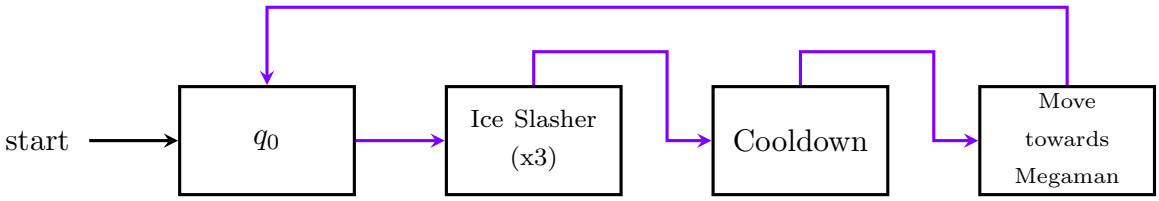

Figure A.19 : Ice Man FSM

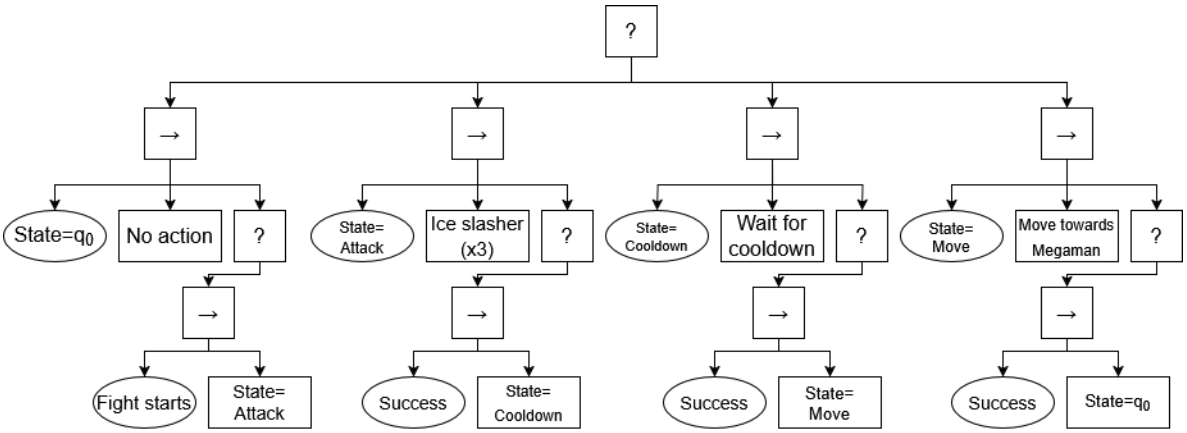

Figure A.20 : Ice Man BT

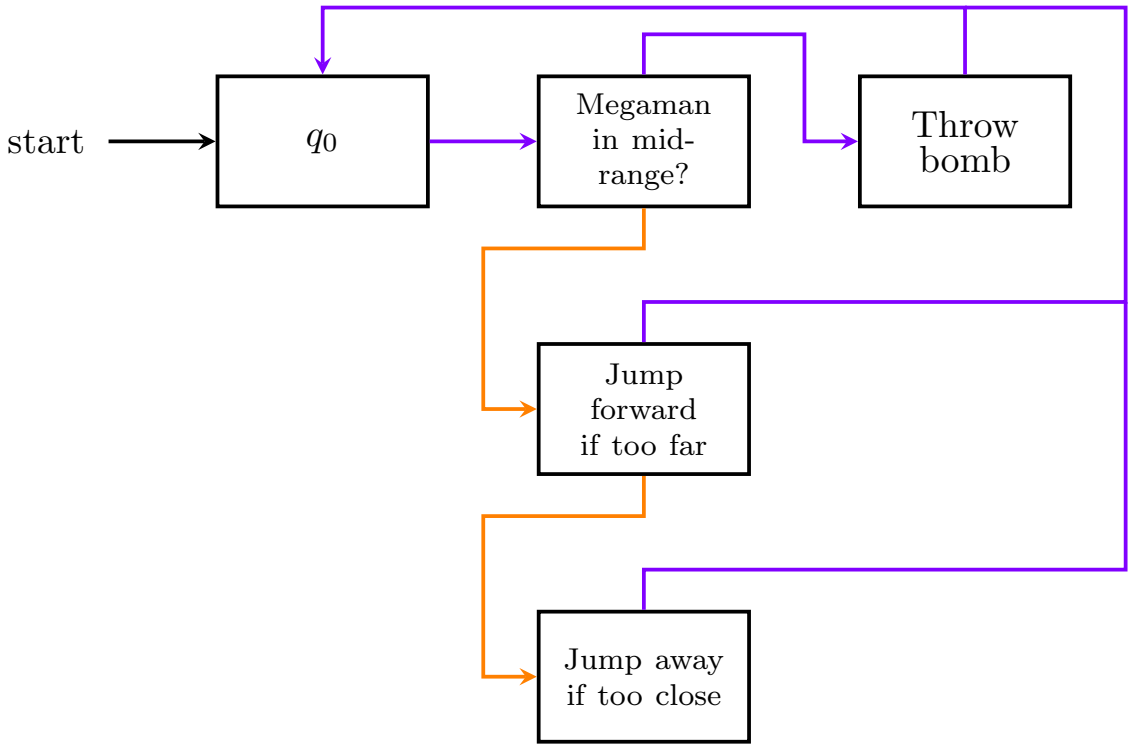

Figure A.21 : Bomb Man FSM

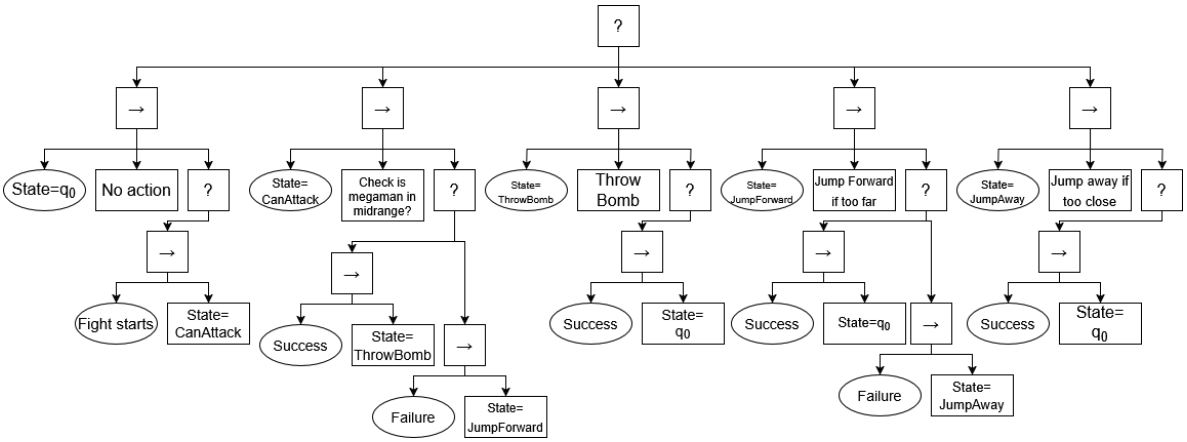

Figure A.22 : Bomb Man BT

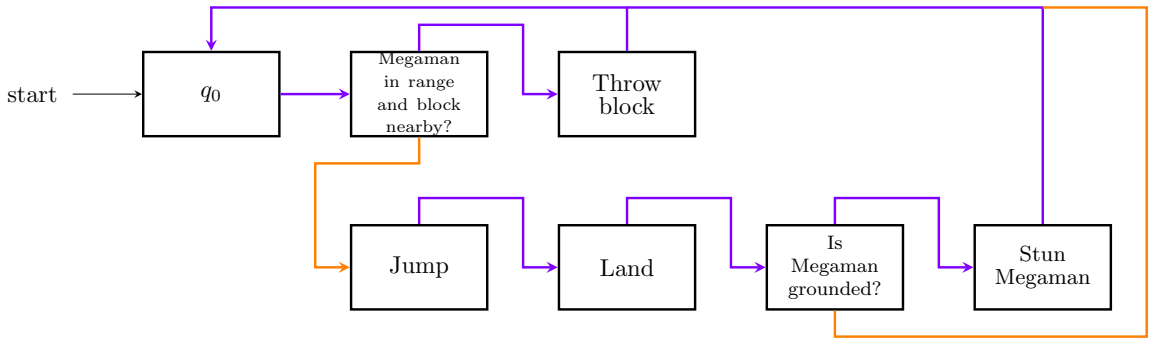

Figure A.23 : Guts Man FSM

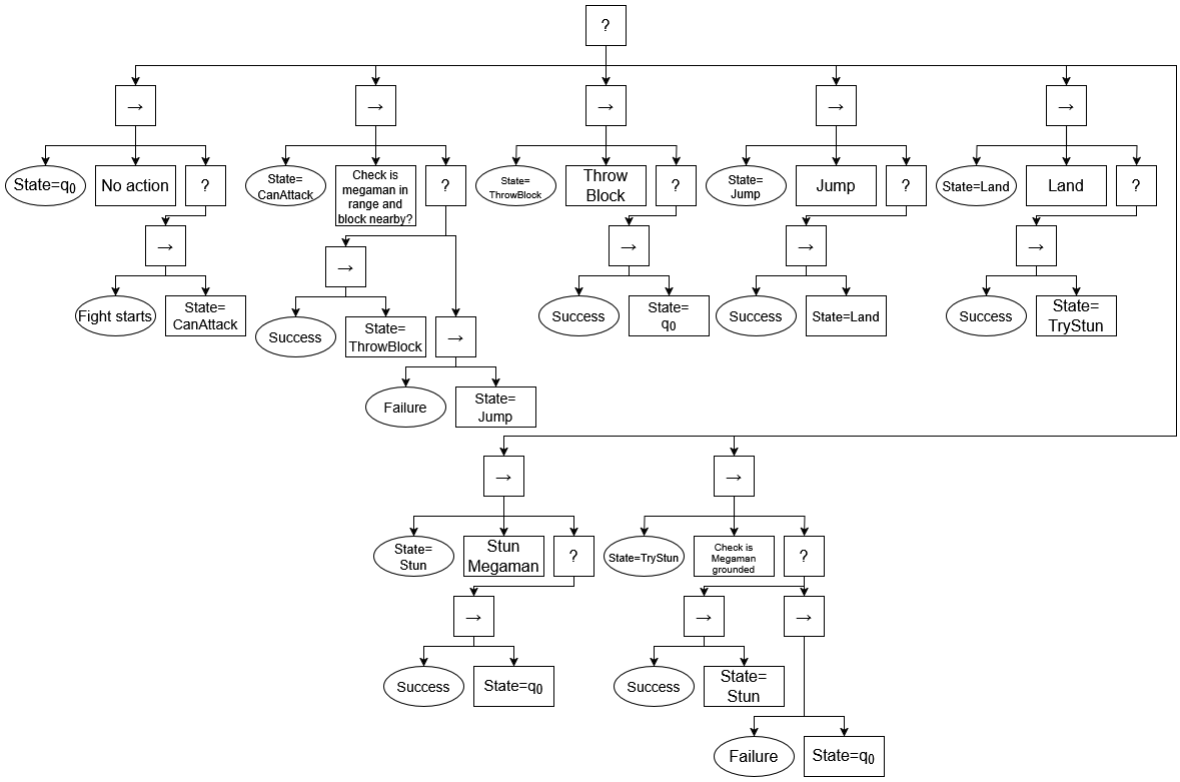

Figure A.24 : Guts Man BT

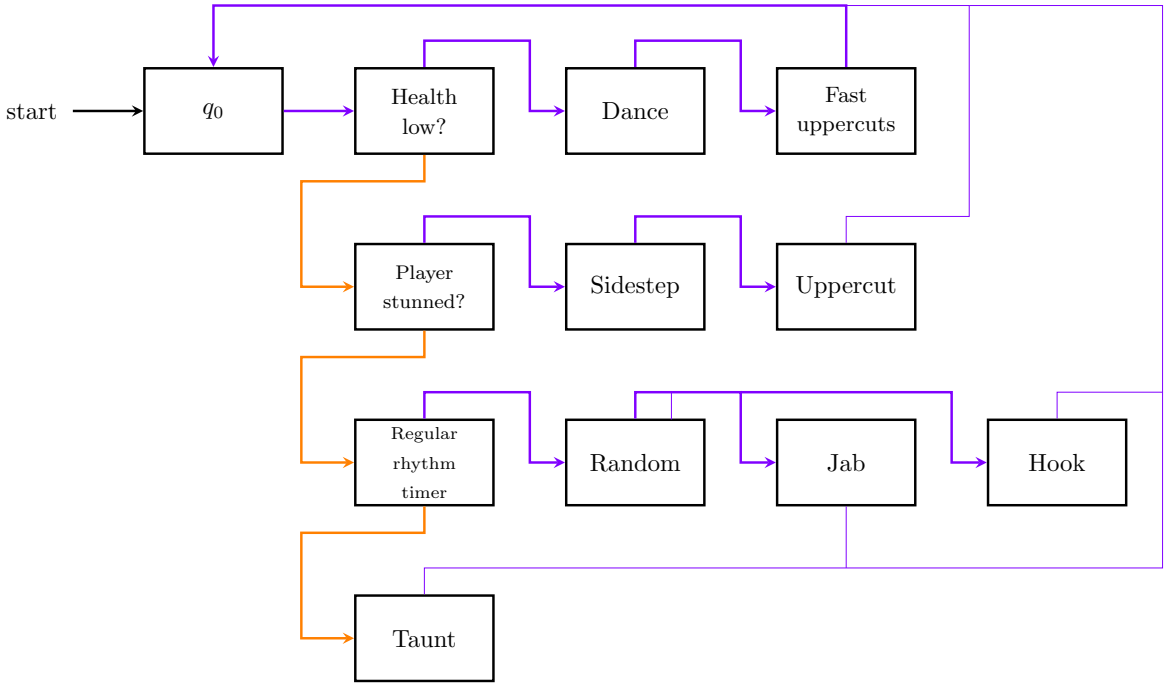

Figure A.25 : Bob Charlie FSM

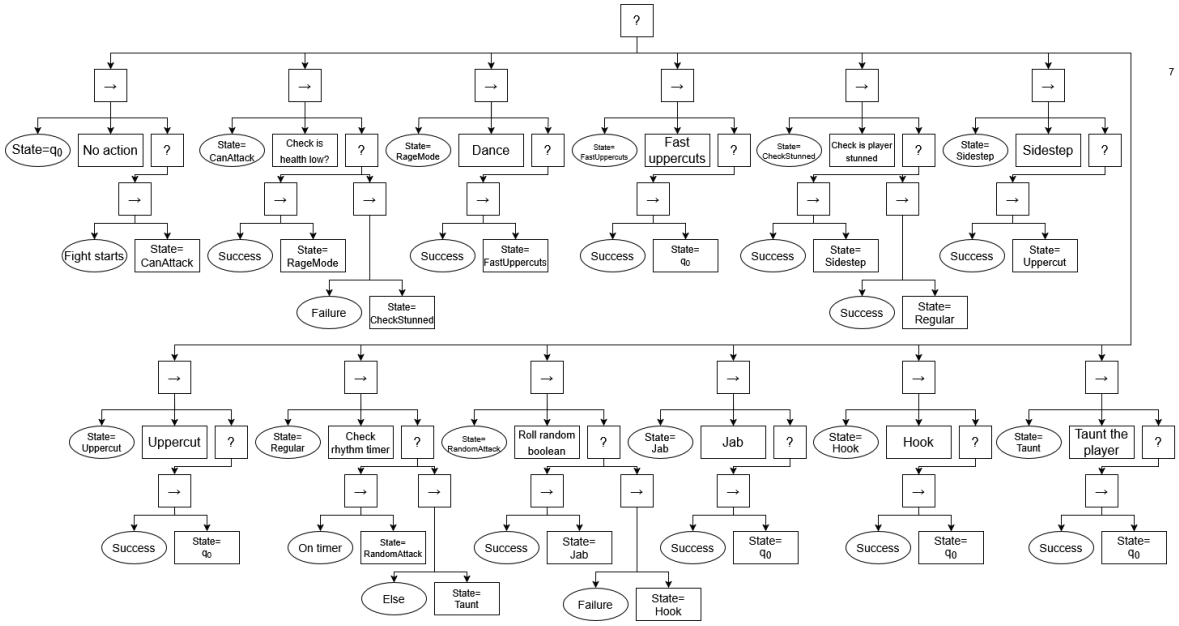

Figure A.26 : Bob Charlie BT

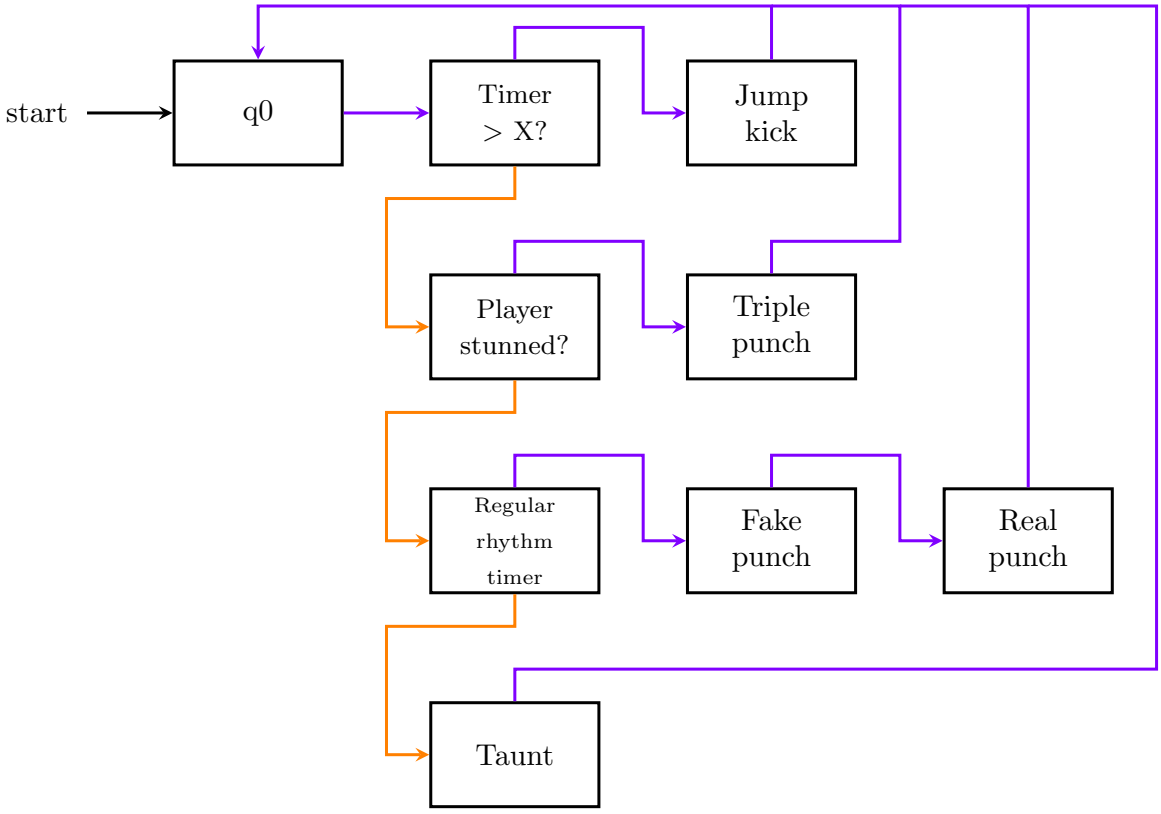

Figure A.27 : Dragon Chan FSM

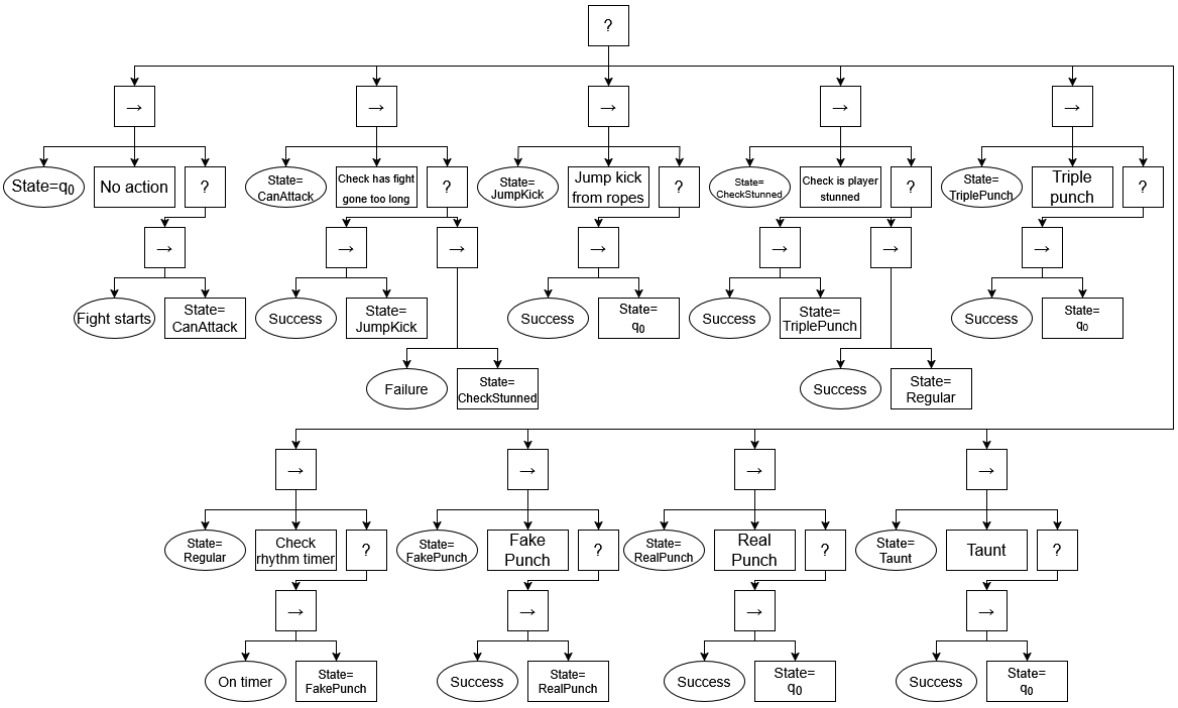

Figure A.28 : Dragon Chan BT

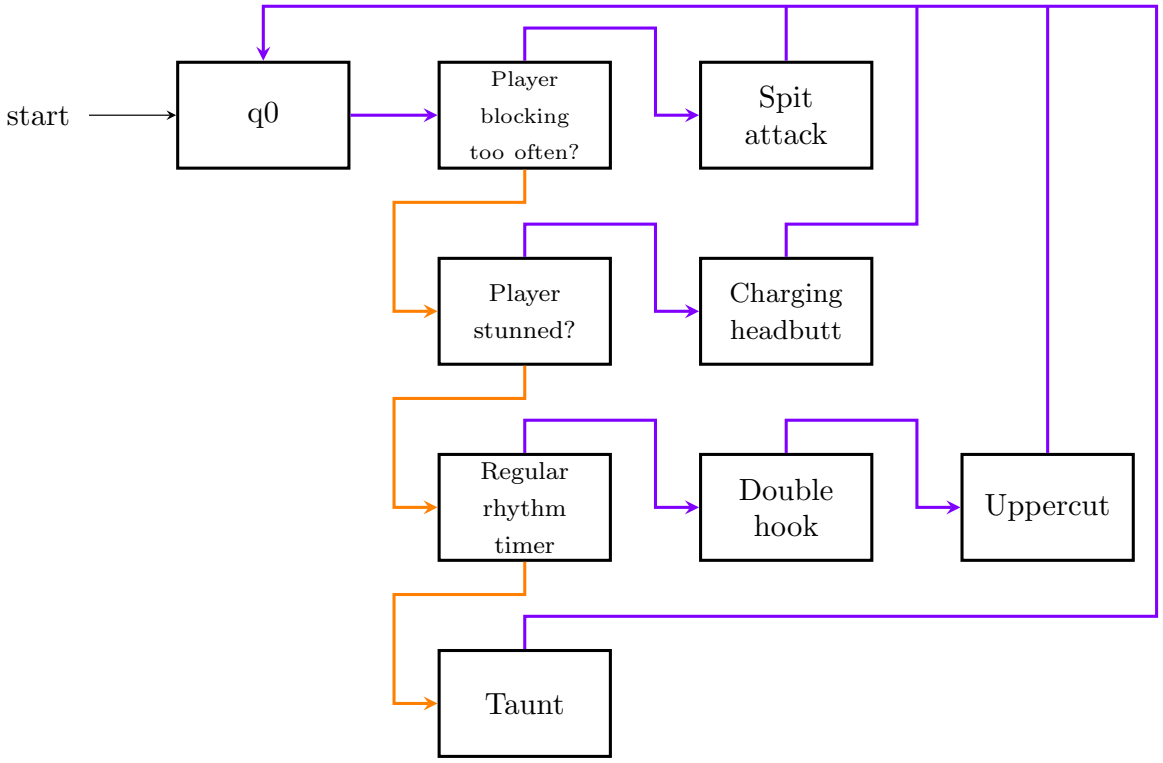

Figure A.29 : Masked Muscle FSM

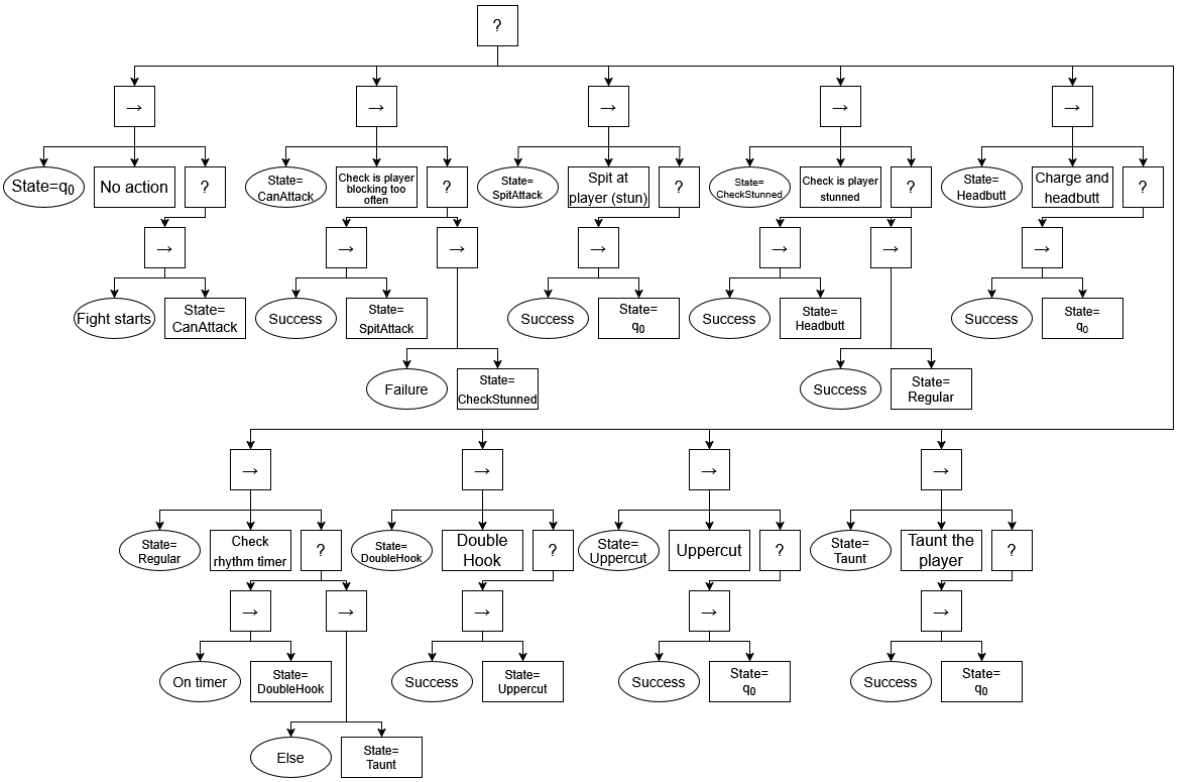

Figure A.30 : Masked Muscle BT

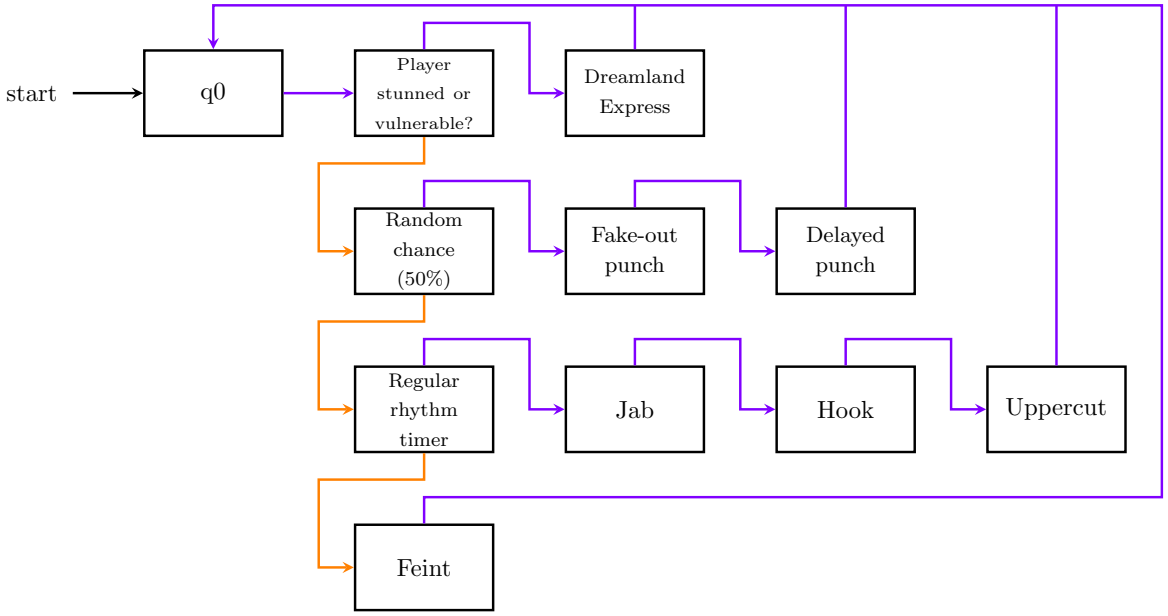

Figure A.31 : Mr Sandman FSM

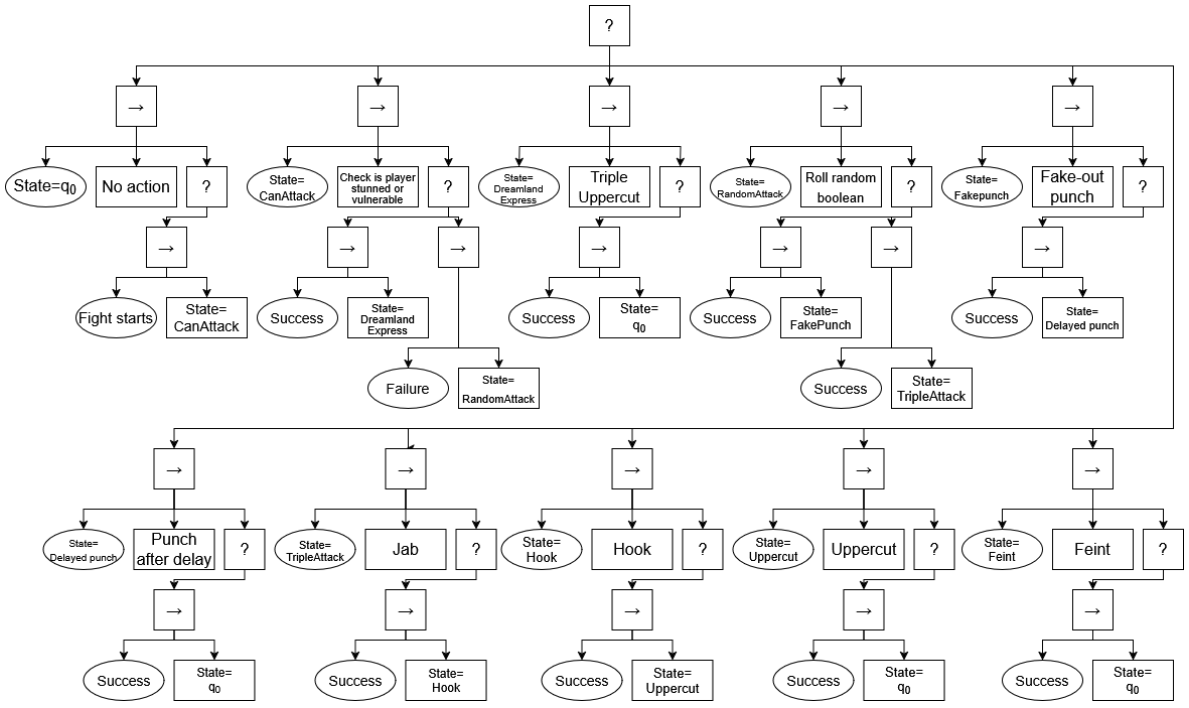

Figure A.32 : Mr Sandman BT

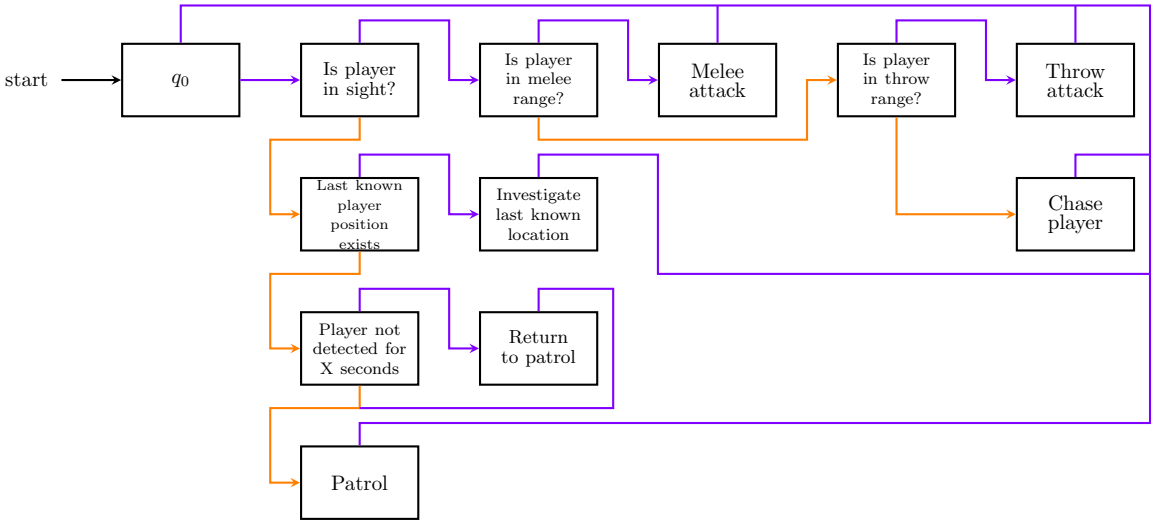

Figure A.33 : Bandit FSM

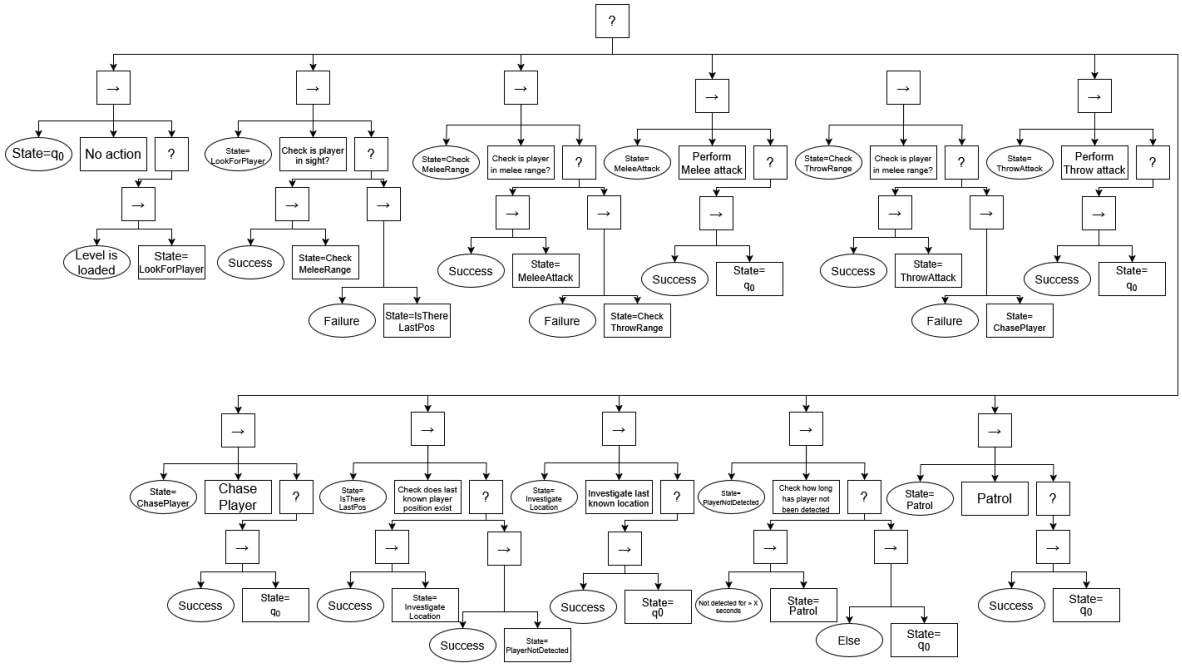

Figure A.34 : Bandit BT

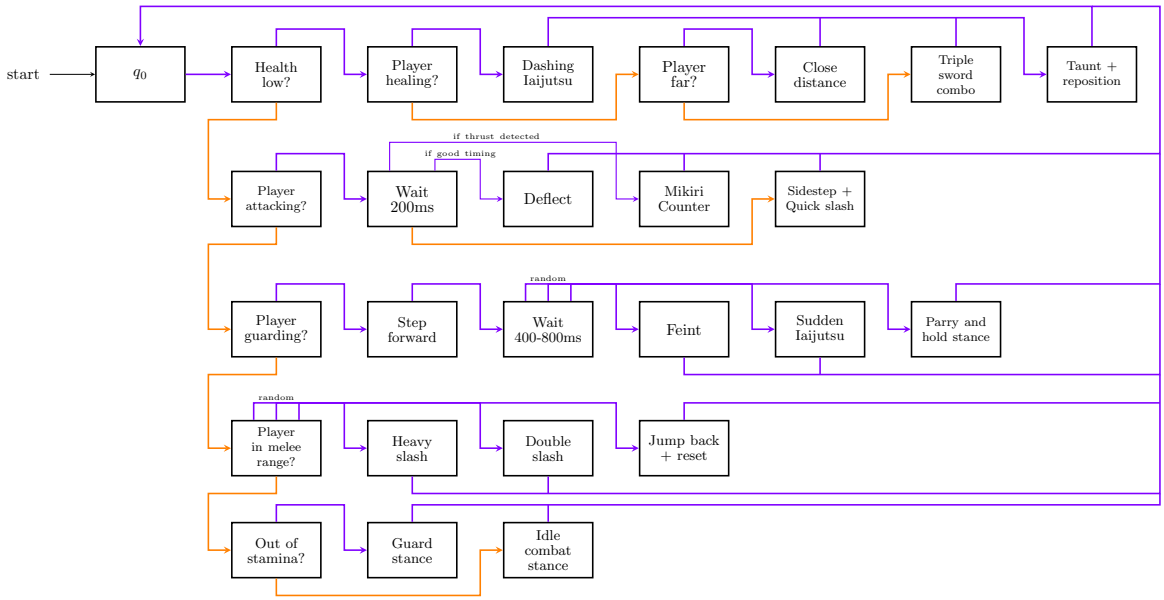

Figure A.35 : Ashina Elite FSM

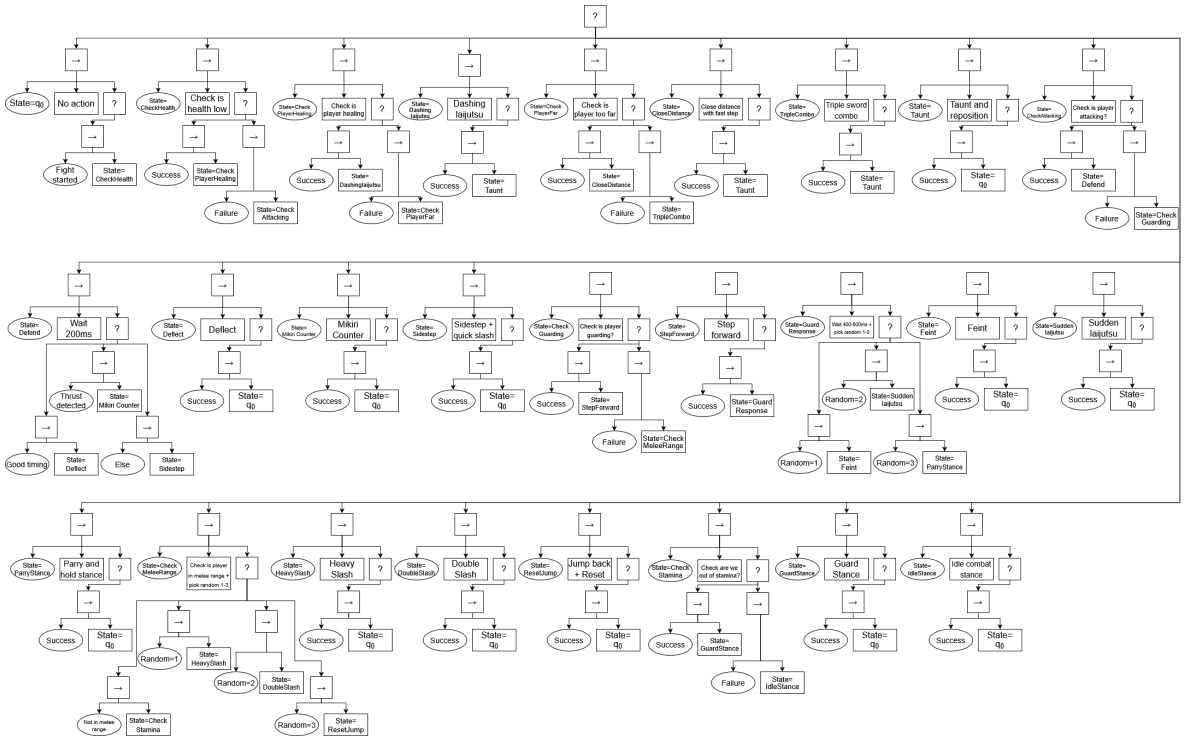

Figure A.36 : Ashina Elite BT