Genetic algorithm for dynamic difficulty AI for game NPCs

Alba Reinders Sánchez
Technical University of Denmark (DTU)
Copenhagen, Denmark
s212729@dtu.dk

Alejandro Valverde Mahou
Technical University of Denmark (DTU)
Copenhagen, Denmark
s212728@dtu.dk

Abstract

We propose the use of genetic algorithms as a valid tool for solving the task of dynamic difficulty adaptability in video games. We also prove its usefulness by developing and testing a prototype where multiple dynamic difficulty algorithms are implemented on the artificial intelligence (AI) of a non-playable character (NPC) for comparison purposes. Likewise, we demonstrate the utility of applying genetic algorithms and that they pose more interest to game developers than traditional approaches, as they adapt to creating a different solution for every player. We believe that this demonstrates the usefulness of artificial intelligence algorithms in modern video games.

Keywords: dynamic difficulty, genetic algorithms, video games, machine learning

1 Introduction

We propose the use of genetic algorithms as a valid tool for solving the task of dynamic difficulty adaptability in video games. We also prove its usefulness by developing and testing a prototype where multiple dynamic difficulty algorithms are implemented on the artificial intelligence (AI) of the non-playable character (NPC) for comparison purposes.

The motivation for this project comes from the idea of bringing a closer state-of-the-art machine learning techniques to video games to allow game developers to create more complex AI mechanics and behaviours.

More specifically, we focus on dynamic difficulty because it can help a wide range of potential players by adapting the difficulty of the game depending on their ability. Machine learning algorithms allow a deeper level of adaptability in contrast to classical approaches, which are usually simplistic.

2 Goals

The main goal of the project is to develop a simple real-time game with dynamic difficulty on the AI of the NPC using genetic algorithms and focusing on player enjoyment. We also aspire to have a game in which each round feels different and has appropriate difficulty implemented in an entertaining and fair way.

We created a prototype consisting of a small game with only a few mechanics, which allows different levels of complexity and difficulty, with the goal of identifying a difference in ability between the players and the different AIs. We wanted the users to be capable of learning to play the game in less than 5 minutes.

With this in mind, we decided to create a 1 versus 1 fighting game in which dynamic difficulty is present on the enemy AI. We avoided other video game genres, such as platform or strategy games, because the playtime required to appreciate a change in difficulty and measure a player's ability is longer. Furthermore, the method of implementing the difficulty is not as straightforward, and the metrics used to evaluate the results are more complex because of the intrinsic design of these genres.

To evaluate whether the goals of the project were met, we performed various testing sessions with users. These sessions provided insights into the game itself, mechanics, and AI behaviour.

The potential issues and drawbacks that can lead to a failed prototype are mostly related to the development of the algorithm. The AI could not be able to take into account the behaviour of the players or learn from the players' gameplay properly, leading to repetitive games with users that are either bored or frustrated. Another possibility is that the game could be too simple to allow for any real difficulty, making it uninteresting.

We address these possible disruptive problems by building the algorithms iteratively, where every step of the deployment is usable. We also performed usability testing with users from an early development stage to ensure the complexity of the game. Additionally, we try multiple approaches on the AIs with tracer bullets to assess the learning capability of each approach.

3 Background

Our work falls under the Dynamic Difficulty Adjustment (DDA) study. DDA aims to solve the problem of adjusting the difficulty of the game to the ability of each individual that plays the game. When a video game is too easy, players become bored; meanwhile, if it is too difficult, players become frustrated. Therefore, game developers must find a balance between these two extremes, and follow a "flow" [2]. Figure 1 show this idea.

Video games with static difficulty levels require players to choose which difficulty would adapt more to their game style before even starting to play. Potentially worsens the game experience if the decision is not appropriate. The DDA solves

this problem by adjusting the difficulty without disturbing or interrupting the player.

DDA approaches vary from game to game, but in general, they consist of simple modifications of the game that make it harder for experienced players and easier for novices.

More advanced DDA approaches focus on using state-of-the-art machine learning algorithms to solve this task with better and more tailored modifications [7]. Some algorithms used for DDA are probabilistic methods, multilayer perceptrons, neural networks, and reinforcement learning.

DDA has been used in a wide range of video games. For example, they can be used for level creation [3] or, as in this project, for enemy behaviour design.

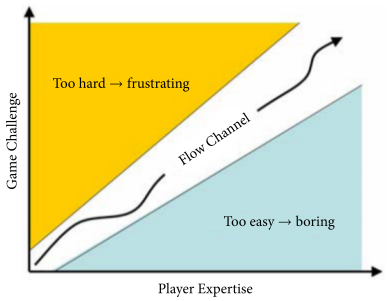

Figure 1. Flow chart[2] proposed by Zohaib[7]

In this project, we used genetic algorithms for the DDA. These algorithms are optimization algorithms that simulate the Darwinian evolutionary process to obtain solutions to a given problem. They generate an initial random population, evaluate the population, and cross the best solutions. The crossing generates new solutions that are a mix of two or more parents and are also slightly mutated to add stochasticity to the algorithm. This process leads to better solutions as the generations pass.

Genetic algorithms are adaptive algorithms that work by adapting and looking for improvements in complex systems and can be applied to games [4]. They were previously used in DDA by Weber *et al.* [6] where they developed a framework that can potentially help implement DDA in any game.

There has been a trend of developing machine learning algorithms, especially using deep learning, to develop AIs that are capable of beating a game or beating human players on it [1, 5]. However, we want to offer solutions where machine learning algorithms are part of the game, making them more interesting to players.

4 The Prototype

The developed prototype consists of a 1 versus 1 fighting video game with three different dynamic difficulty methods implemented on the AI of the NPC. The game is created using *Unity v2021.3.11f1*.

4.1 The Game

The game consists of a closed battle arena where players have to defeat an enemy controlled by the AI until either the AI or the player runs out of lives. The main components of the game are shown in Figure 2. The player controls the blue cat (left) and the AI controls the orange lizard (right). Both can move and perform three possible actions: *Attack* , *Block* , and *Dash* . Successful landing of an attack decreases one health point. Each character has five health points, and once one runs out of the health points, it loses one life. The number of lives depends on the game mode. All actions consume stamina, and if a character is below the required threshold, it cannot perform any action.

The game has four screens: a title screen, where the player selects the game mode; the main screen, where the battle takes place; and the game over screen, which shows the battle result and the loading screen for the genetic algorithm to create the new AI behaviour. For visualization of the screens, refer to Appendix B Game screens.

Each character is defined by two state machines: movement state machine and action state machine. They restrict and control the character's possibilities and current state. These two state machines are shown in Figure 3 and 4.

4.2 The AI

The AI of the game is defined by eight parameters that represent the probability of moving from one state to another in both state machines and in each frame. The parameters are: `IdleToFollow`, `IdleToWander`, `FollowToIdle`, `WanderToIdle`, `IdleToAttack`, `IdleToDash`, `IdleToBlock` and `BlockToIdle`. These are all transitions that can be done on purpose by the bot, as shown in Figure 3 and 4.

The three different dynamic difficulty methods for the AI of NPC are as follows:

1. *Genetic algorithm*:
   It simulates the player results in previous rounds in an attempt to generate enemies that perform similarly to player gameplay.
   A warm-up session is required, in which the user plays against a predefined bot. Subsequently, the loading screen appears while five generations of the genetic algorithm are trained.
   The algorithm minimizes the difference in health points between the AI bot and the player. We decided to use five generations, as it was the best trade-off between execution time and accuracy of results.
   Once the genetic algorithm finishes, the player fights against the best bot.
2. *Classical approach*:
   It uses the classical method of adapting the game difficulty in fighting video games, which changes the characteristics of the AI, but not its behaviour.

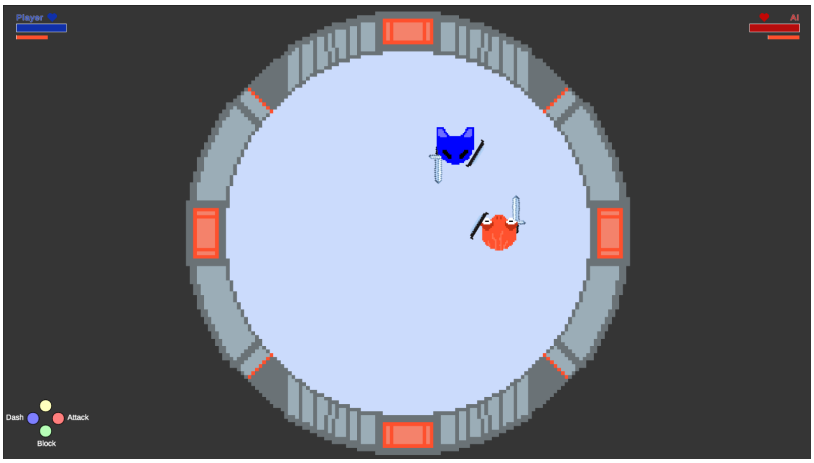

Figure 2. Main screen of the game prototype. The bottom left corner displays the possible actions and the corresponding buttons. The player and AI states are on the top left and right corners respectively (lives, health points and stamina).

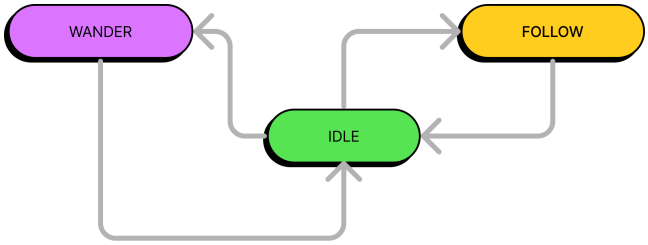

Figure 3. Movement state machine.

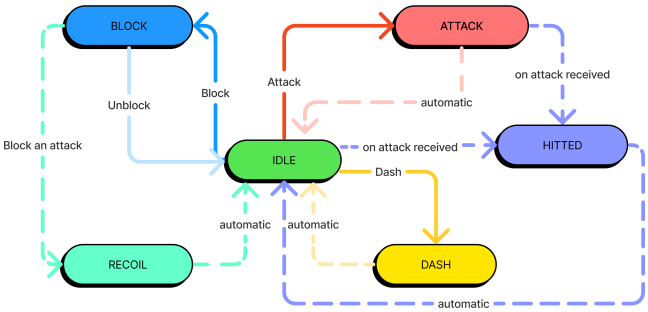

Figure 4. Actions state machine. Continuous lines represent actions made on purpose, and discontinuous lines represent automatic states changes

We modified the *speed*, *attack strength* and *stamina recovery* of the enemy depending on whether the player was winning or losing.

3. *Ranking system*:
   It consists of a pool of randomly generated AIs ordered by the number of victories obtained in a previous offline tournament, where all bots played against each other five times.
   We created 50 different bots. The user starts playing with the bot in the middle (25); then, if the user wins the match, it goes up in the ranking and down in the other cases.

The last two methods were developed to compare the results obtained using the genetic algorithm.

5 Development Methodology

We decided on the following agile development methodology, where we initially designed a long-term plan, but we only defined the specific tasks one week ahead. We held a short meeting at the start of the week to create and divide the next week's tasks.

We organized the full development process into five phases:

1. *Discover and exploration*: The first phase of the development was focused on deciding and defining the topic to develop. This involved research papers and group discussions. We destined 2 weeks for it.
2. *Game development*: The idea behind this phase was to have a working game prototype where we could iterate and create our AI on. We spent 2 weeks on it.
3. *AI development*: Once we had a working game, we focused on creating a working AI which we modified to obtain better performance. In this phase, most of the tracer bullets occurred. This was the longest phase, with 6 weeks of development.
4. *Test and iterate*: This phase was completely centred on user testing, to evaluate and asses if we met our goals, as well as analyse the testing results. We used 3 weeks on it.
5. *Final touches*: The last phase was only 1 week long and was used to finish the report and evaluate additional results.

For further explanation and visualization of the project timeline, refer to Appendix C Planning structure.

The main tracer bullets were the different algorithm implementations that were developed and tested. The first tracer bullet was on how to code character actions. We tried a simple approach based on `if-statements`, but soon we realized it would make more sense in the long run to use a state machine because it is simpler to know what actions are available at each moment based on the current state.

The second tracer bullet was centred on whether the game state information should be used to code the bot behaviour. We developed a probabilistic approach where the bot did not use the game state, and a bot that used it. By testing with users, we found that the bot that did not use any game state was more interesting to users than the others.

Our last tracer bullet helped us choose the algorithm to use on the AI. We attempted both the reinforcement learning and genetic algorithm approaches. Reinforcement learning requires longer user interaction to correctly adapt to players' needs; therefore, we decided to use genetic algorithms.

6 Test Methodology

To obtain insights and results from our prototype, we conducted two test sessions.

The first test session took place during the development of the game and the basic behaviour of the AI, with the purpose of analysing the complexity and design of the game, as well as how the AI interacts with the player.

This test helped us define the final game mechanics and led us to change the interaction between players and the game. This also led to UX improvements to facilitate the correct understanding of the game and its actions. Some examples of changes include the introduction of a stamina bar to reduce the possibility of constantly and exclusively attacking behaviour observed in multiple players. Alternatively, the decision to implement a countdown before each round, as players requested it after being caught not prepared to start the round.

The second testing session focused more on the AI and its complexity, especially to compare the results and difficulty perceptions of the three different methods.

The results of this test lead to our analysis and results, which are further explained in Section 7 Test Results, Analysis and Takeaways.

The method followed in both testing sessions was letting the users play the game with only basic explanations and then fill a short survey. We decided not to explain how the methods worked, as we believe they could bias the results of the survey. With the goal of getting more testers, we offered a small reward in the shape of candies. The average testing session lasted approximately seven minutes. All players used the same controller to play the game.

Doing multiple testing sessions, with different objectives, allowed us to further focus our project around players, as well as make the necessary changes to overcome the errors, test multiple tracer bullets, and assess if we were able to meet our predefined goals.

7 Test Results, Analysis and Takeaways

An important remark is the comparability of the three AI methods. Although all of them try to implement dynamic difficulty, they are not truly comparable, as the genetic algorithm creates the AI bot in real time, while the other modes are prepared beforehand and can be modified and fine-tuned. Furthermore, it is not trivial to adequately implement the ranking system. We decided to generate the AI bots completely randomly, but to perform a fair comparison against the genetic algorithm, it should have been trained. The problem with this is the lack of a metric to decide when to stop the training, making the creation of the ranking system a whole new problem to solve.

We are aware of this unfairness in the comparison, but we strongly believe it provides context, and it is of interest to have a starting solution against which we can compare.

As previously mentioned, the testing session was divided into two parts, where users played the game first and then answered the survey.

We collected multiple data variables from the player matches. The most relevant one was who won the round (the player or the bot) and how many health points the winner had at the end. In Figure 5 we display the percentage of rounds won by the players in the three modes (Mode 1: genetic algorithm, Mode 2: classical approach, and Mode 3: ranking system). This figure contextualizes and summarizes the general analysis. We strive towards a player's average win rate between 60% and 75%. We can observe that the ranking system has a very high win rate, making it a less interesting approach because players can win fairly easily. The other two modes are within our desired margins and require a more detailed analysis.

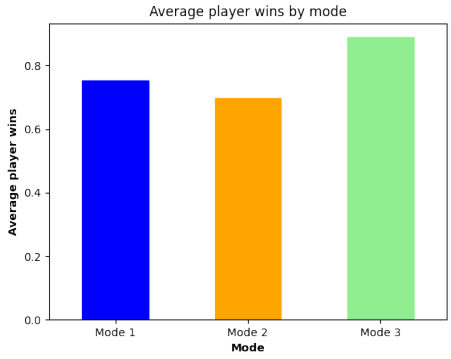

Figure 5. Average player's wins by mode. Mode 1: Genetic algorithm, Mode 2: Classical approach and Mode 3: Ranking system.

Regarding the answers from the survey, we asked users about both the general game prototype and their gaming ability. We highlight the game ability that users believe they have as an interesting question, as it leads to meaningful insights. Our data are skewed for users who played against the genetic algorithm. This is shown in Figures 6 and 7, and we believe that this affects the results, as shown in Figure 5 where the win rate is higher in mode 1, where more experienced users play.

Further details on the results obtained from the survey can be found in Appendix D Survey results.

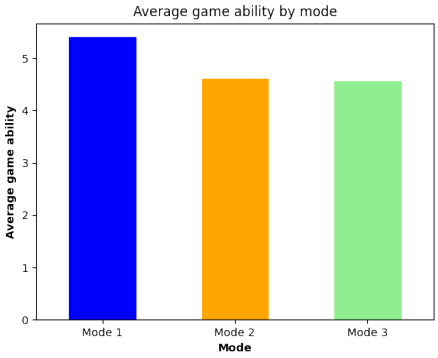

Figure 6. Average player's ability by mode. Mode 1: Genetic algorithm, Mode 2: Classical approach and Mode 3: Ranking system.

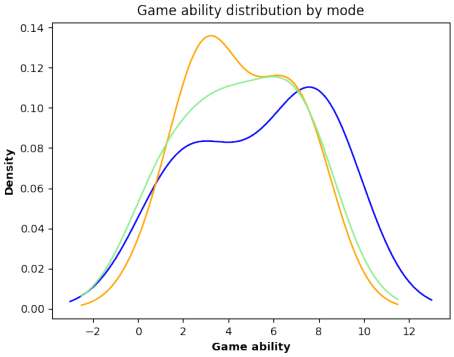

Figure 7. Game ability distribution by mode. Mode 1: Blue line, Mode 2: Yellow and Mode 3: Green.

Then, we analysed whether the appreciation of the general difficulty of the game per mode followed any relation. In Figure 8, we see that users think the classical approach is the most difficult, followed by the genetic algorithm and the ranking system. The results may indicate that the classical approach is a better approach to dynamic difficulty, as it is both more balanced, with an average difficulty very close to our ideal value, three out of five, and easier to implement to any video game. But, as we have stated, we have data unbalancing, which could mean that because more experienced users played against the genetic algorithm, they did not feel the game was that difficult, unlike the users who played against the classical approach, which seemed to be less experienced and, therefore, felt the game was harder.

We observe in Figure 9 that the classical approach distribution follows our desired shape, a normal around the value "3". Meanwhile, the genetic algorithm distribution is less sharp but also around the "3" value.

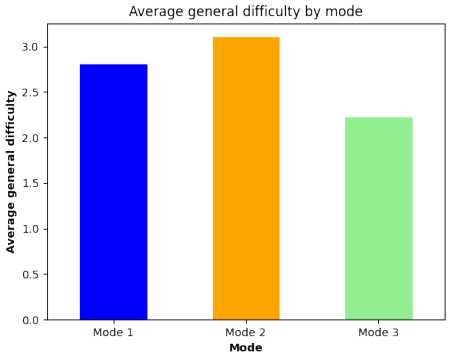

Figure 8. Average general difficulty by mode. Mode 1: Genetic algorithm, Mode 2: Classical approach and Mode 3: Ranking system.

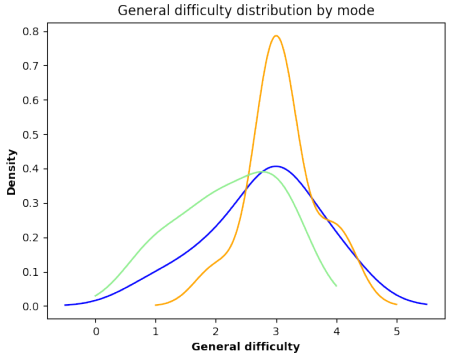

Figure 9. General difficulty distribution by mode. Mode 1: Blue line, Mode 2: Yellow and Mode 3: Green.

8 Discussion

In this project, we developed and tested a fighting game prototype using multiple methods of dynamic difficulty with end-users. We achieved an entertaining game in which each round felt different owing to a probabilistic state machine.

During the testing sessions of our prototype, we were able to assess the various problems that our initial game had

regarding both user experience and artificial intelligence behaviour.

We compared multiple dynamic difficulty approaches to test and analyse if genetic algorithms are a viable way to solve the problem of adapting to each individual gaming ability. We conclude that although the classical approach was better suited to our game, the data obtained were biased, and we could not formulate any meaningful conclusions.

Despite this drawback, we show the utility of applying genetic algorithms and that they pose more interest to game developers than traditional approaches, as they adapt to creating a different solution for every player, and we believe that games with more difficulty could create more interesting results.

Future work on this topic will focus on testing the approach on more complex interactive systems and comparing them with other machine learning approaches to prove the usefulness of modern artificial intelligence techniques in modern industry video games.

References

[1] Kai Arulkumaran, Antoine Cully, and Julian Togelius. 2019. AlphaStar: An Evolutionary Computation Perspective. http://arxiv.org/abs/1902.01724 cite arxiv:1902.01724.

[2] Mihaly Csikszentmihalyi. 1991. *Flow: The Psychology of Optimal Experience.* Harper Perennial, New York, NY. http://www.amazon.com/gp/product/0060920432/ref=si3_rdr_bb_product/104-4616565-4570345

[3] Martin Jennings-Teats, Gillian Smith, and Noah Wardrip-Fruin. 2010. Polymorph: Dynamic Difficulty Adjustment through Level Generation. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games* (Monterey, California) *(PCGames '10)*. Association for Computing Machinery, New York, NY, USA, Article 11, 4 pages. https://doi.org/10.1145/1814256.1814267

[4] Robert E. Marks. 2002. *Playing Games with Genetic Algorithms.* Physica-Verlag HD, Heidelberg, 31–44. https://doi.org/10.1007/978-3-7908-1784-3_2

[5] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* 529, 7587 (Jan. 2016), 484–489. https://doi.org/10.1038/nature16961

[6] Matheus Weber and Pollyana Notargiacomo. 2020. Dynamic Difficulty Adjustment in Digital Games Using Genetic Algorithms. In *2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. 62–70. https://doi.org/10.1109/SBGames51465.2020.00019

[7] Mohammad Zohaib. 2018. Dynamic Difficulty Adjustment (DDA) in Computer Games: A Review. *Advances in Human-Computer Interaction* 2018 (2018), 1–12. https://doi.org/10.1155/2018/5681652

A Contributions overview

The scientific paper was written by Alba Reinders Sánchez (ARS) and Alejandro Valverde Mahou (AVM). The distribution of its writing was the following:

0. Abstract: co-authors ARS and AVM
1. Introduction: co-authors ARS and AVM
2. Goals: ARS
3. Background: AVM
4. The Prototype: ARS
5. Development Methodology: AVM
6. Test Methodology: AVM
7. Test Results, Analysis and Takeaways: ARS
8. Discussion: co-authors ARS and AVM

The code was implemented by both authors using pair programming, where ARS focussed more on the game development and the testing. Meanwhile, AVM focussed on the AI development and data analysis. All authors have knowledge of all steps of the development and have equal contribution.

B Game screens

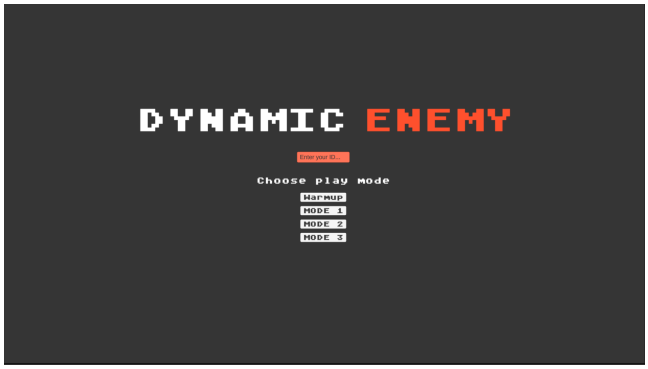

Figure 10. Game Screen 1: Title screen

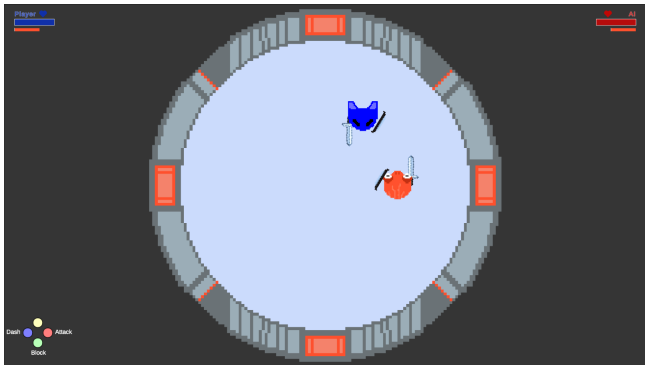

Figure 11. Game Screen 2: Game Screen

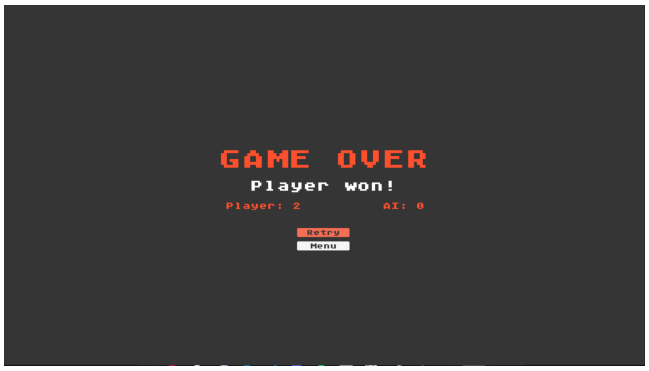

Figure 12. Game Screen 3: Game over screen

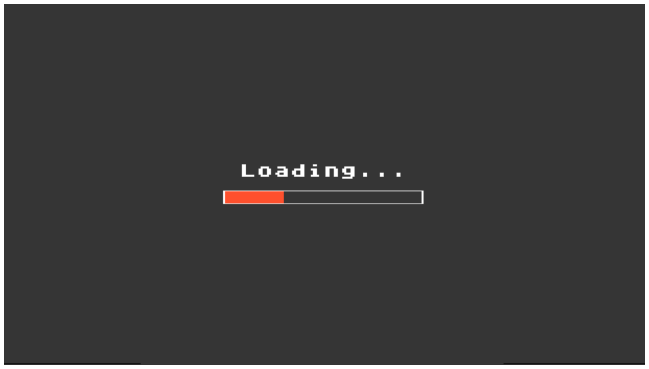

Figure 13. Game Screen 4: Loading screen

C Planning structure

Figure 14 displays the timeline of the project. The phases refer to the divisions in time and the sections refer to the type of work. We divided the project in 5 different sections:

1. *Research*: Tasks that involved researching papers and looking up tutorials.
2. *Game Design*: Task that involved developing or improving the game.
3. *AI Design*: Tasks that involved developing or improving the AI.
4. *AI Testing*: Tasks that involved testing the AI with users or ourselves.
5. *Writing*: Tasks that involved making presentations or reports.

D Survey results

The following figures show all the results obtained from the survey.

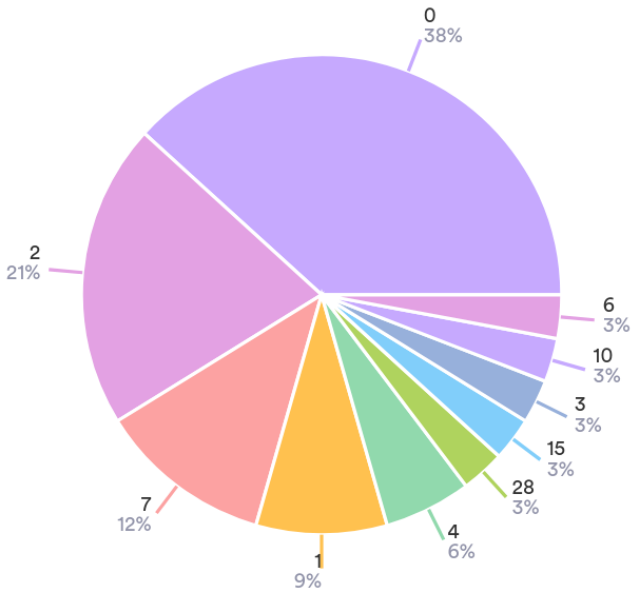

Figure 15. How many hours do you play a week?

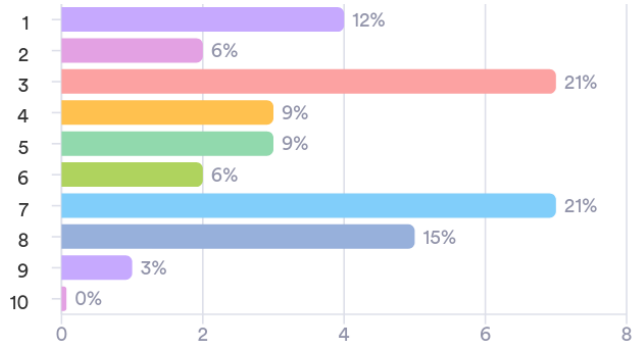

Figure 17. How much ability do you believe to have playing games?

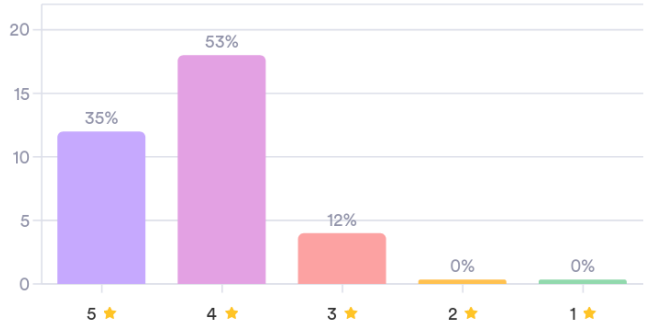

Figure 18. Did you enjoy the game?

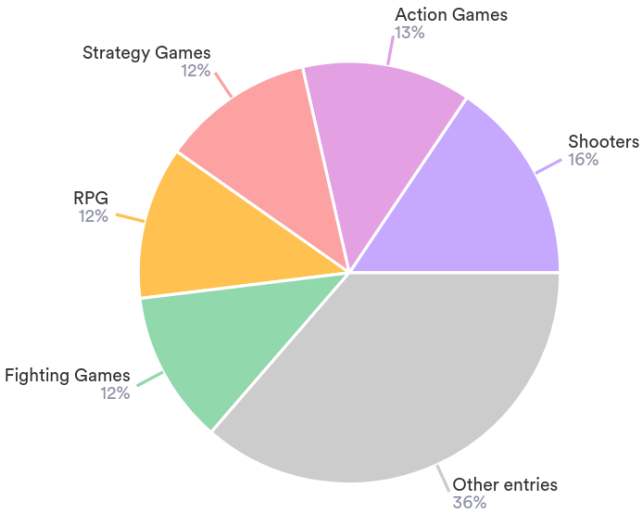

Figure 16. What genre of games do you usually play?

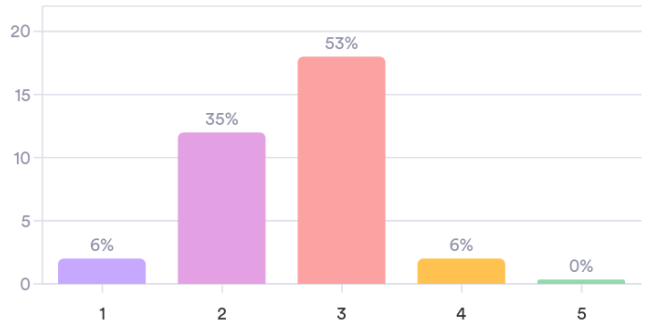

Figure 19. How long did you find the game?

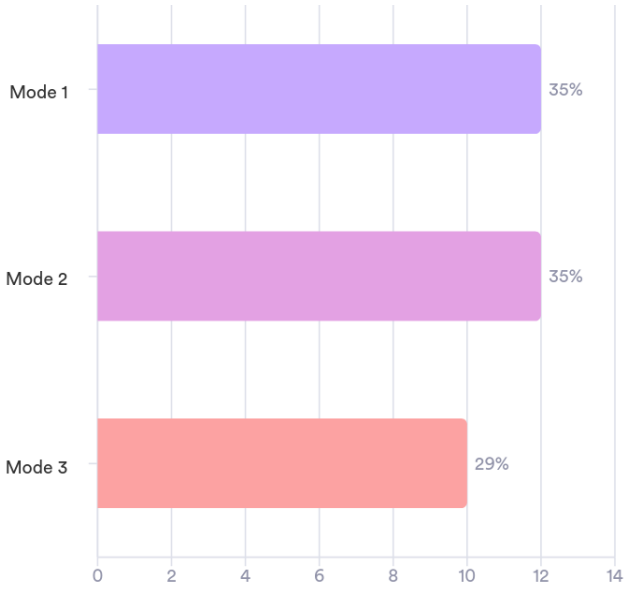

Figure 20. Which game mode did you play in?

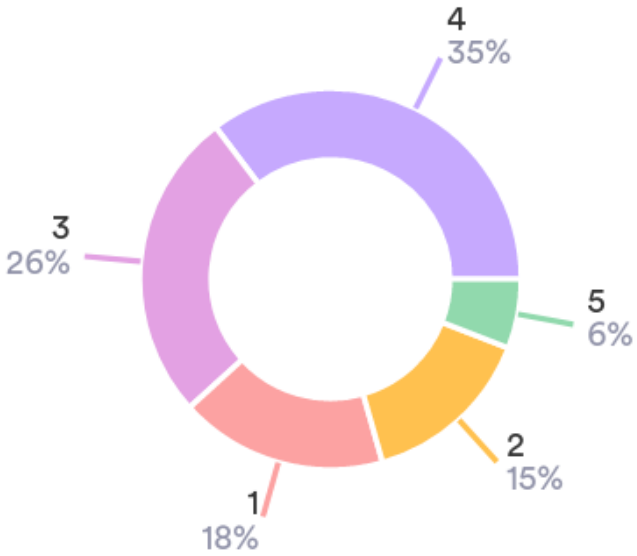

Figure 22. How difficult was to defend against the bot?

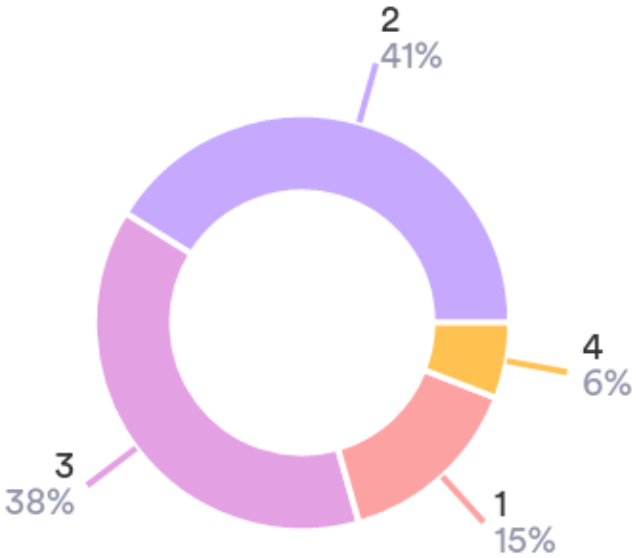

Figure 21. How difficult was to attack the bot?

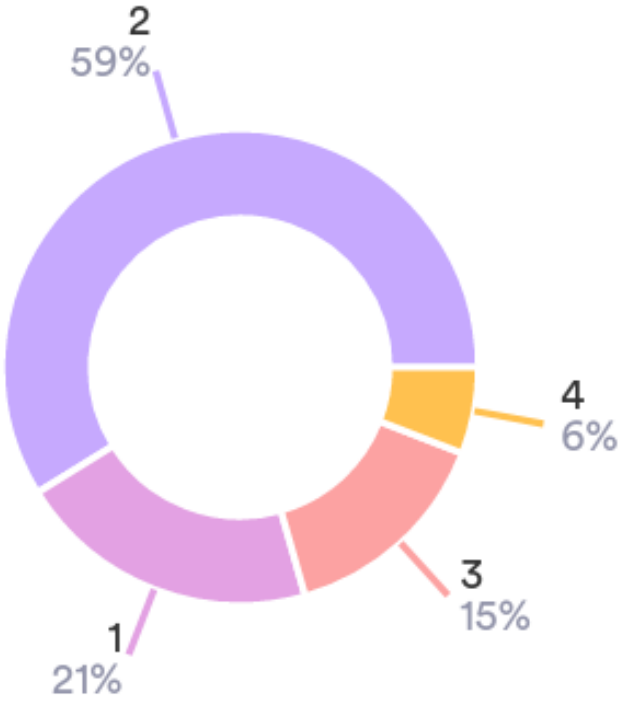

Figure 23. How difficult was to catch the bot?

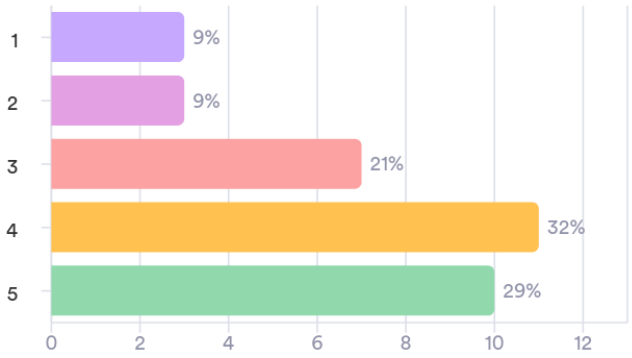

Figure 24. Did you notice any change or adaptability of the bot?

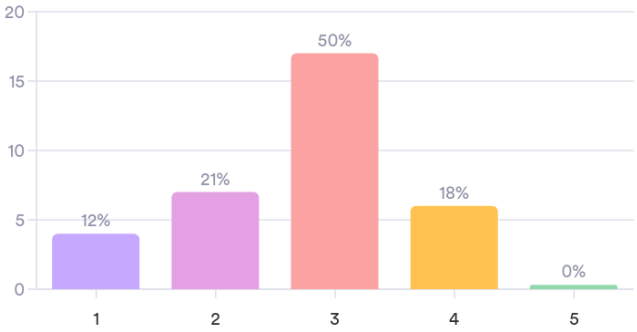

Figure 25. How difficult was to play against the bot?

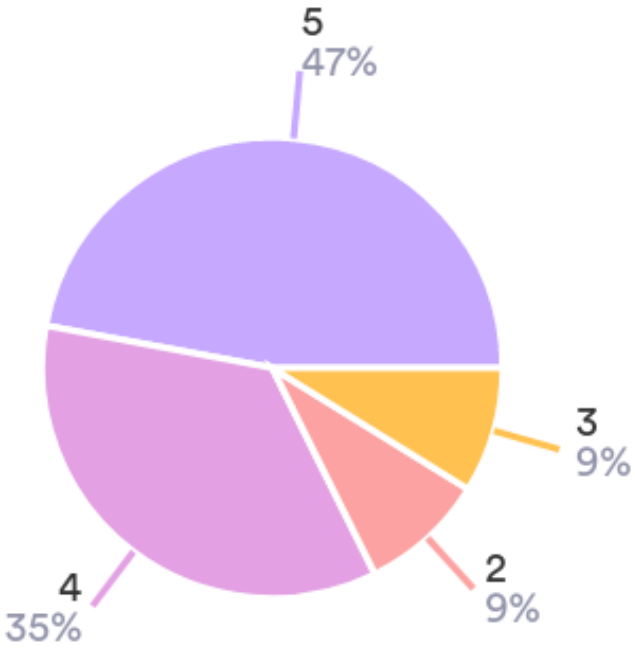

Figure 26. Would you like to continue playing the game?

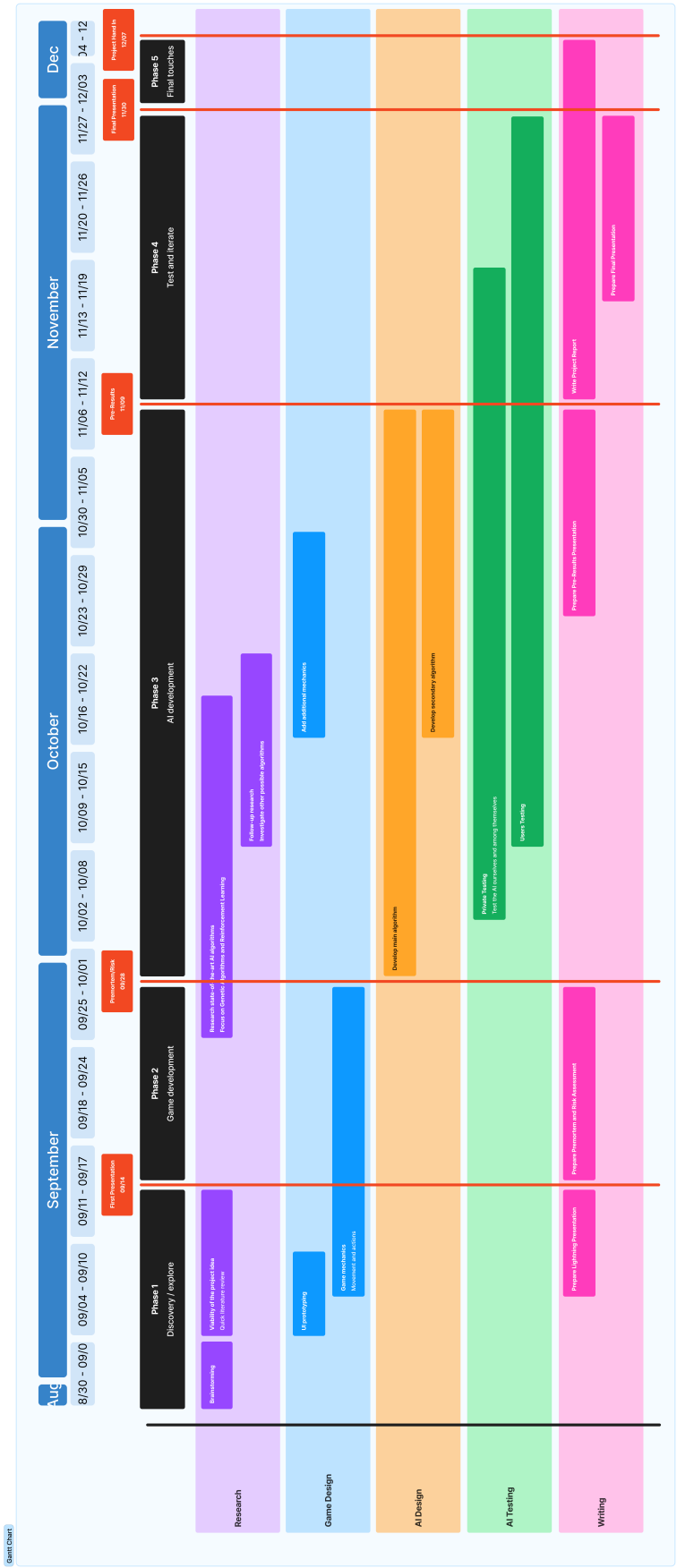

Figure 14. Gantt chart of the project planning