Final Report Senior Project

รายโครงงานทางวิศวกรรม

เรื่อง

การสร้างศัตรูระดับบอสโดยอัตโนมัติ

Automatic Boss Enemy Generation in Game

โดย

ศุภสิน รุ้งสิทธิกุล รหัสนิสิต 5931069721

อาจารย์ที่ปรึกษา

ผศ. ดร. วิษณุ โคตรจรัส

รายงานนี้เป็นส่วนหนึ่งของวิชา 2110499 โครงการวิศวกรรมคอมพิวเตอร์พื้นฐาน

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ประจำปีการศึกษา 2562

Table of Contents

บทคัดย่อ

เกมประเภทแอ็คชั่นเป็นประเภทของเกมที่ได้รับความนิยมอย่างสูง ในเกมประเภทแอ็คชั่นนั้นผู้เล่นจะควบคุมตัวละครต่อสู้กับศัตรูด้วยความสามารถและอาวุธที่มี บ่อยครั้งเมื่อจบด่านหรือถึงตอนสำคัญของเนื้อเรื่อง ผู้เล่นต้องเผชิญศัตรูระดับบอสและพิชิตมันเพื่อดำเนินเกมต่อ ศัตรูระดับบอสนั้นสร้างความท้าทายให้ผู้เล่นพิชิต ให้ประสบกรณ์การต่อสู้ที่สนุกและ เป็นหนึ่งส่วนประกอบสำคัญในหลาย ๆ เกมประเภทแอ็คชั่น

ในการสร้างศัตรูระดับบอสตัวหนึ่งต้องลงแรงและเวลามาก การสร้างบอสแบบอัตโนมัติสามารถช่วยแบ่งเบาภาระหรือเป็นประโยชน์ต่อการสร้างบอสให้ผู้พัฒนาได้ โครงงานนี้จึงมีความต้องการสร้างเกม พัฒนา และ ทดสอบวิธีการสร้างบอสขึ้นมาแบบอัตโนมัติด้วยขั้นตอนวิธีเชิงพันธุกรรม เพื่อที่กระบวนการที่ใช้ในโครงงานนี้จะสามารถเป็นประโยชน์ให้กับการสร้างบอสแบบอัตโนมัติในเกมแอคชั่นอื่น ๆ ได้

บอสของเกมในโครงงานนี้มีลำดับของพฤติกรรมหลายลำดับ และแต่ละลำดับก็มีการกระทำที่จะกระทำตามลำดับนั้นๆ สำหรับการกระทำต่างๆของบอส ได้ถูกเขียนไว้โดยมีค่าตัวแปรต่างๆและใช้ขั้นตอนวิธีเชิงพันธุกรรมในการสุ่มสร้างและหาลำดับการกระทำและกระทำที่มีค่าตัวแปรดีเหมาะสมตามคุณสมบัติ สำหรับค่าความเหมาะสมของบอสแต่ละตัวในขั้นตอนวิธีเชิงพันธุกรรมของโครงงานนี้ได้มาจากการการนำบอสไปทดลองต่อสู้กับผู้เล่นปัญญาประดิษฐ์ และนำข้อมูลจากการต่อสู้มาประมวลผลด้วยฟังก์ชันค่าคาดหมาย ขั้นตอนวิธีเชิงพันธุกรรมนั้นได้ผลลัพธ์เมื่อดำเนินจนถึงจุดอิ่มตัว ผลลัพธ์บอสที่ได้เมื่อพิจารณาตัวอย่างจากบอสห้าตัวที่มีค่าความเหมาะสมสูงสุด มีพฤติกรรมที่น่าพึงพอใจ แต่ความหลากหลายของพฤติกรรมยังไม่มากนัก

Abstract

Action game genre is one of the most popular game genres. In an action game, a player typically controls a character and then battles enemies with the character's skills and weapons. At the end of a level or important point of the story, the player often encounters a boss enemy and must defeat it for progression. Boss enemies pose challenges for the player to overcome, provide enjoyable battle experiences and are the important elements in many action games.

In order to create a boss enemy, a lot of efforts and time have to be invested. Automatic boss generation can alleviate the workload or be useful to the developers in creating bosses. The aim of this project is to create a game of which automatic boss generation is explored and tested using genetic algorithm. The technique developed by this project can then be used as a guide for automatic boss enemy generation in other action games.

Bosses in this project's game have several attack sequences. Each attack sequence has it order of actions to perform. Each action is fixed behavior with parameters. Genetic algorithm was used to randomly generate actions with random parameters and find appropriate attack sequences and actions. For each boss fitness value, a player AI played against the boss and obtained the battle data which was then used to calculate fitness. Genetic algorithm returned result bosses when convergence criteria was met. Five bosses with the highest fitness value from the result was inspected. Their behavior was satisfactory. However, they were not very distinctive from one another.

1. Introduction

1.1 Motivation and Significance

Action game is a game genre that emphasizes physical challenges such as reflex and reaction. It is a vast genre covering all games that involve physical challenges. Action game has long been one of the most popular game genres. [1][2]

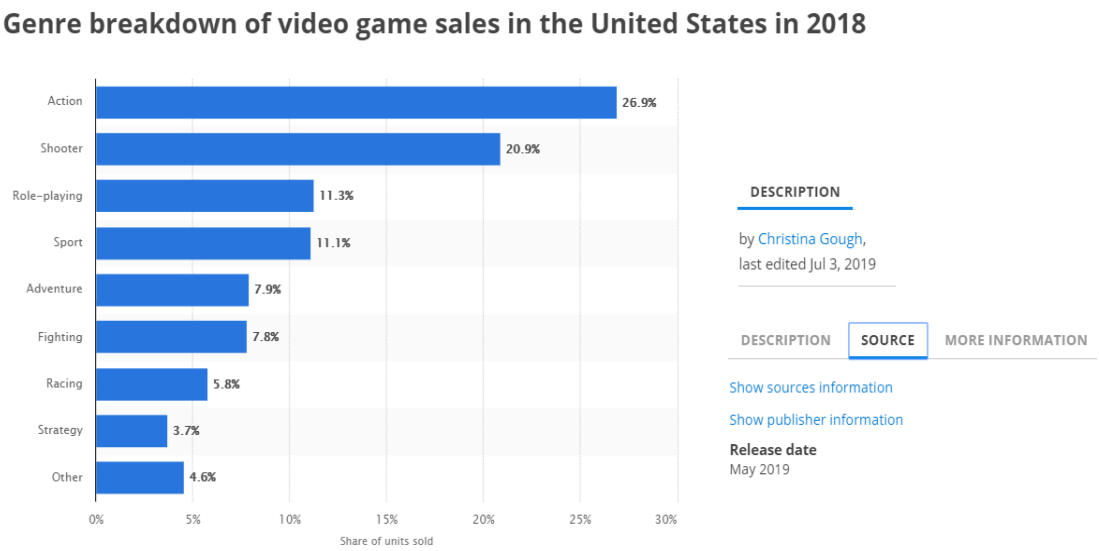


Figure 1. Genre breakdown of video game sales in the United States in 2018. [3]

In an action game, a player typically controls a character often in the form of a protagonist or avatar and then battle enemies with their skills as well as weapons and other tools at their disposal. At the end of a level or important point of the story, the player often encounters a boss enemy and must defeat it to proceed to the next level or to progress the story. Boss enemy poses challenges for the player to overcome, provides enjoyable battle experiences and is one of the important elements in many action games. [4]

In order to create a boss enemy, a lot of work, time and efforts must be invested by designers and developers. Automatic boss generation could alleviate the workload if many boss enemies are needed for a game.

This project's aim is to create a game and boss enemy generation by using genetic algorithm for generating appropriate bosses. A boss is considered appropriate if it can be defeated and it takes a reasonable amount of time to be defeated. Furthermore, it must not just stand still, it should also attack and provide a fight to pose challenges to players.

1.2 Objective

The aim of this project is to create a game with automatic boss generation, using genetic algorithm.

1.3 Scope

- The game is 2D gameplay with 3D graphics. 3D models and 3D perspective camera are used but movement is restricted in 2D dimensions.
- The environment or the level is fixed to a limited space on a square platform without any obstacle.
- Actions that the player can do and the player's parameters such as maximum health and attack power are fixed.

1.4 Expected Benefits

The technique developed by this project can then be used as a case study for other automatic boss enemy generation projects.

2. Related Theories and Related Research

2.1 Related Theories

2.1.1 Action game

[5] Action games include physical challenges. They may also incorporate puzzles, races, and a variety of conflict challenges, typically among a small number of characters. Action games often contain simple economic challenges as well, usually involving collection objects. They seldom include strategic or conceptual challenges.

2.1.2 Genetic Algorithm

[6] A genetic algorithm is an evolutionary computation method. Most methods called "Genetic Algorithms" have at least the following elements in common: populations of chromosomes, selection according to fitness, crossover to produce new offspring, and mutation.

Population

The process begins with a set of individuals. A set of every individual is called a population. Each individual is a solution to the problem and is characterized by a set of parameters (variables) known as Genes. Genes are joined into a string to form a Chromosome (solution).

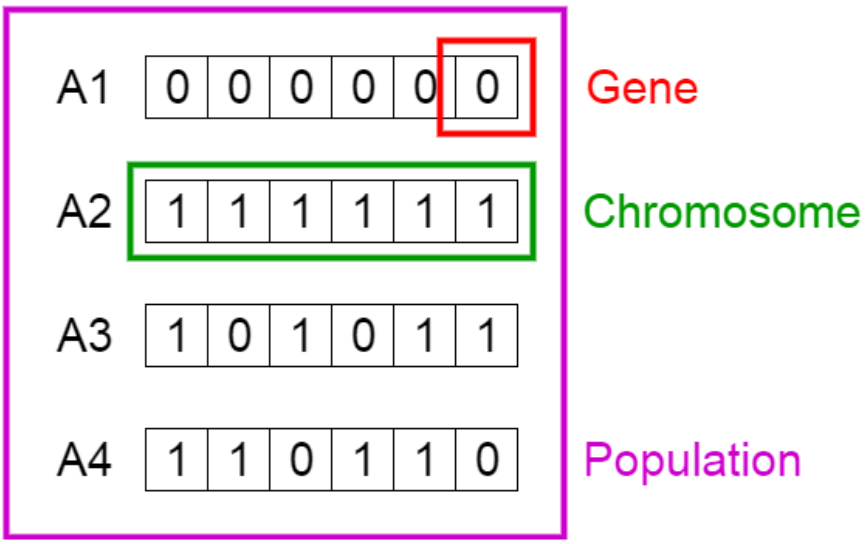


Figure 2. Population, Chromosomes and Genes. [7]

Selection

In each generation, the fitness of every individual in the population is evaluated with fitness function(s) which tell how good the individual is. The probability that an individual will be selected for reproduction is based on its fitness score.

Crossover

This step generates the next generation population from the population left from the selection. Parents, which can be more than two, are selected from the population to generate children by exchanging the genes of parents among themselves. This process is repeated until an appropriate population size reached.

Mutation

For each new child, it has a low random probability that some of its genes can be subjected to a mutation that changes the gene randomly. [7] [8]

2.2 Related Research

2.2.1 Research about automatic enemy generation

Pichit Promsutipong [9] described enemies in Agent Description Language for 2D game. The enemy data was then mined for relations that frequently showed. Then relations were chosen randomly by probability proportional to its frequency and translated back in form of functions and parameters and inserted into the agent skeleton. The enemy agents were then decorated by filling missing parts and removing unnecessary parts. Acceptable enemy behavior pattern were behavior patterns that could be defeated or avoided by players. Enemy agents were then tested by a player AI and measured by number of hits that the player AI took and number of hits that the player AI avoided.

For our project, the idea of inserting functions and parameters to agent skeleton was adopted to model boss enemies. The idea of using player AI to play with enemies was adopted for data gathering for fitness function.

2.2.2 Research about boss enemy

Theodore Agriogianis [10] observed game bosses and formulated ideas from them to build a better understanding of the boss concept and create bosses in his game.

The interesting part that relates to ourthis project is boss fight: A boss must be beatable by the player and it should be defeated by using player skill. About boss fights, the player should not be completely overwhelmed by the boss. Boss attacks often have some sort of sign that tells the player that it is attacking and it gives the player time to respond to the attack.

3. Project Development

3.1 The Game

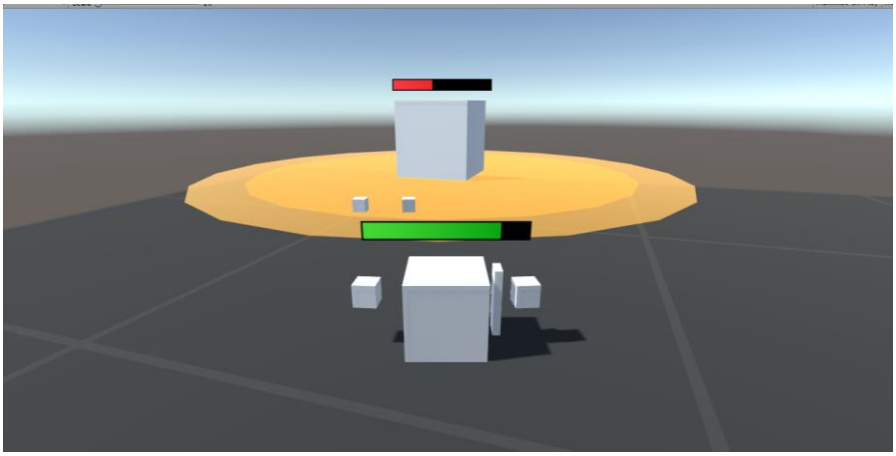


Figure 3. A screenshot from the game.

The game created in this project was modeled after multiplayer games, Final Fantasy XIV and Dragalia Lost. Players controlled a character with limited choice of actions and avoided boss attacks which came in the form of geometric colored area indicator on the floor (such as circle and square). These indicators were called AoE (Area of Effect). It dealt damages to players colliding with it when its timer ran out. Our game removed multiplayer levels and multiplayer mechanics since it was for one player.

Figure 4. An example screenshot of "Dragalia Lost" combat and attack indicators.

Figure 5. An example screenshot from of "Final Fantasy XIV" attack indicators.

3.1.1 Player

Possible actions for player were:

- Move: move forward, move backward, move left and move right.
- Rotate: rotate left and rotate right.
- Attack: melee attack (deal 5 damage) and range attack (spawning 2 projectiles that deal 1 damage each).

When the player attacked, the player would not be able to move or rotate for an amount of time. Melee attacks dealt more damage but if the boss stayed too far away, ranged attack was more reliable. Player character has 100 HP (Health Point).

3.1.2 Boss

Each boss had several attack sequences. While fighting, a sequence was chosen randomly. Each attack sequence had its own order of actions. After the last action of a sequence was performed, the boss randomly chose an attack sequence to perform again. Every boss has 100 HP.

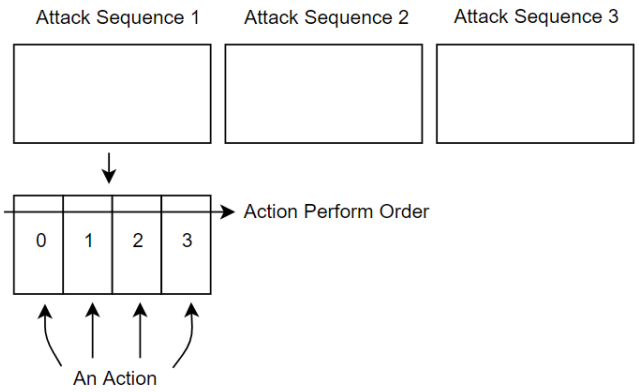


Figure 6. An illustration of attack sequences.

Actions for bosses were categorized to attack action, movement action, and blank action. Attack actions created AoE (Area of Effect). AoE dealt damage to the player in contact with it when it was activated (a countdown was displayed on screen). Movement action moved the boss to a designated position on the arena or to the player (with an offset). Each type of attack action or movement action had fixed behavior with parameters as shown below:

Common attack action parameters:

Parameter	Value Range	Description
DelayBefore	[0-2]	Idle time before performing the attack.
DelayAfter	[0-2]	Idle time after performing the attack.
AoeTimer	[3-7]	Timer for AoE spawned by the attack.
Damage	[5,20]	Amount of player health that will be reduced if player is hit by AoE spawned from this action.

Table 1: Common attack action parameters.

Attack actions with their description and additional parameters:

Attack Name	Description	Parameter	Value Range
Attack at boss	Create an AoE with type AoeType at boss position.	AoeType	[Square,Circle]
Attack at player	Create an AoE with type AoeType at player position.	AoeType	[Square,Circle]
Attack toward player	Create a square AoE that points toward the player from the boss position with offset distance and minimum length.	AoeType	[Square]
		plusDistance	[-10,10]
		minLength	[5,10]
Attack at a coordinate	Create an AoE with type AoeType at the designated coordinate.	AoeType	[Square,Circle]
		Coordinate	[Coordinates]

Table 2: Attack action description and additional parameters.

Common movement action parameters:

Parameter	Value Range	Description
DelayBefore	[0-2]	Idle time before performing the movement.
DelayAfter	[0-2]	Idle time after performing the movement.
TimeToMove	[0-2]	Time used to perform movement

Table 3: Common movement action parameters.

Movement actions with their description and additional parameters:

Movement Name	Description	Parameter	Value Range
Move to Coordinate	Move to designated coordinate.	Coordinate	[Coordinates]
Move toward player	Move toward the player with offset distance.	distanceOffset	[-10,10]

Table 4: Movement action description and additional parameters.

Additional parameters for each AoeType parameter:

AoeType	Parameter	Value Range	Description
Square	xSize	[4-30]	Width of the square AoE.
	zSize	[4-30]	Height of the square AoE.
	rotation	[0-180]	How much the Aoe would rotate from initial setting around y-axis and in clockwise direction.
Circle	diameter	[4,30]	Diameter of the circle AoE.

Table 5: Additional parameters for each AoeType parameter.

Possible coordinate values are shown in Figure 7. (use none to use position parameters instead):

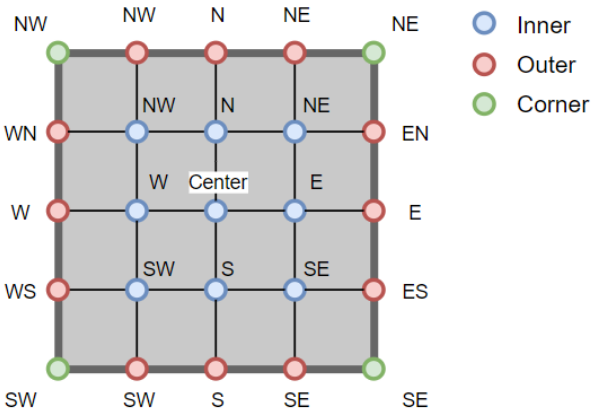


Figure 7. Coordinate positions and names.

if coordinate is none, parameters below are used for a random coordinate instead.

Parameter	Value Range
xPos	[-15,15]
zPos	[-15,15]

Table 6: Parameters to be used in case of coordinate parameter is none.

Note that arena size was 30x30 unit.

3.2 Boss generation

An action of a boss was represented as a gene and an attack sequence as a block in action list. Blank action was utilized for reducing the number of actions in an attack sequence.

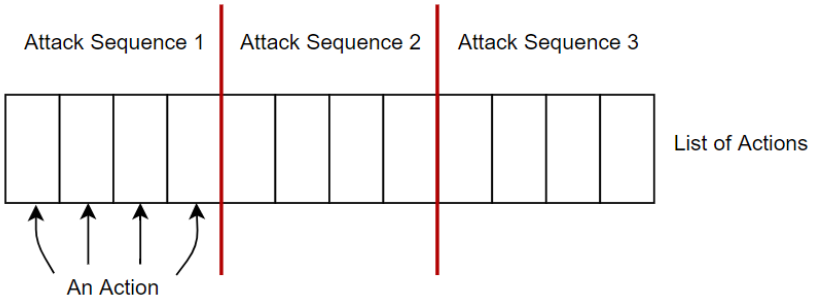


Figure 8. An illustration list of actions and attack sequences of bosses.

3.2.1 Selection

Number of fittest individuals survived for each generation was calculated from survive rate and populations count. For a boss fitness value, a player AI would play against the boss and recorded the battle data. Data from the battle were time that the player AI took to defeat the boss or be defeated by the boss, player AI health left and time that the player AI was on AoE(s) or boss's attack uptime. If the player AI health was 0, the player AI was defeated by the boss. The fitness functions were:

Expected time for beating the boss:

$$F_T = weight_T \frac{expect_T - |expect_T - T|}{expect_T}$$

Expected attack uptime percentage:

$$UpT_{\%} = \frac{T - UpT}{T}, D_{UpT\%} = 0.5 - |0.5 - expect_{UpT_{\%}}|$$

$$F_{UpT_{\%}} = weight_{UpT_{\%}} \frac{expect_{UpT_{\%}} - |expect_{UpT_{\%}} - UpT_{\%}|}{D_{UpT_{\%}}}$$

Expected player health left:

$$D_{Hp} = \frac{MaxHP}{2} - |\frac{MaxHP}{2} - expect_{HP}|$$

$$F_{HP} = weight_{HP} \frac{expect_{HP} - |expect_{HP} - UpT|}{Divider_{Hp}},$$

Boss was beaten:

$$F_B = weight_B \; ; if \; boss \; is \; beaeten$$
$$F_B = 0 \qquad ; player \; is \; defeated \; by \; the \; boss$$

The overall fitness value was the summation of each fitness.

3.2.2 Crossover

After the selection, crossover replenished the population by randomly picking a pair of parents and produced offspring until population count was the same as in the previous generation. For crossover, 3-point crossover was used. Two crossover points were picked randomly and genes between the two points were swapped.

3.2.3 Mutation

For mutation, each gene or each action had a possibility to be mutated to a randomly new action with the mutation rate.

3.2.3 Convergence

Convergence criteria used in this project was: the difference between the average fitness value of the current generation and the average fitness value of the previous generation became less than an acceptable average difference value.

3.3 Player AI

3.3.1 Player AI by Reinforcement Learning

This project attempted to implement player AI by using Unity ML-Agents which was a machine learning toolkit for Unity. The reinforcement learning algorithm this project attempted to use for player AI was Policy Optimization (PPO) from ML-Agents. Used observations and agent rewards functions were as follows:

Observation:

- Player position
- Player rotation.
- Boss position.
- Current AoE touching count.
- Angle between player forward vector and boss position.
- AoE touching count and the value indicating whether the player character detector was out of boundary for each detector. Detector is a collision detector that attached to player character at an offset position from player for each one.

Agent reward functions:

- -0.0005 for each step.
- -0.01 for each current AoE touching count for each step.
- +0.05 per 1 boss health loss HP.
- -0.01 per 1 player health loss .
- Angle between player forward vector and boss position reward for each step, if angle <= 90 degree, reward is positive else if angle > 90 degree, reward is negative.

  size of reward is $(0.005)\frac{|angle-90|}{90}$.
- +1.0 if the agent beat the boss.
- -1.0 if the agent is defeated.

After some training and tuning, the player AI agent performance could not meet the expectation. This project therefore changed the player AI approach by using rule-based player AI instead.

3.3.2 Player AI by rule-based

The player AI was programmed to always rotate itself to face toward boss and attack if it did not collide with any AoE. For player AI movement, its behaviors were separated into 2 cases:

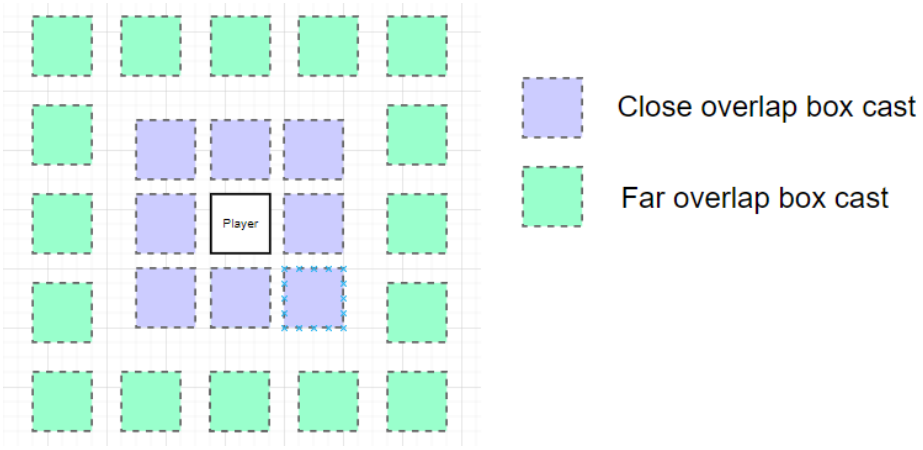


Figure 9. Player AI box casts.

1. Player AI was not colliding with any AoE attack: the AI would cast overlap boxes close to itself which had the same size as the player and detected whether the position at the casted box collided with a number of AoE. Then, if there was a safe position(s) that did

not collide with any AoE, the AI would move to the safe position closest to the boss. If there was no safe position, the AI would stay at the same position.

2. Player AI was colliding with any number of AoE: the AI would cast overlap boxes with more spacing between each box and combined the result with close casts. Then, the AI would try to move to the closest safe position and if there was more than one position with the same distance to the player AI, the AI would choose the one closest to the boss.

4. Results

4.1. Boss Generation Results

The run configuration is shown in the table 7. The algorithm started at population which generated bosses randomly with attack sequence size and size of actions in an attack sequence as in configuration. Then the algorithm performed selection, crossover and mutation until it met convergence criteria.

Name	Value	Description
Number of game instances	80	Number of game instances and total population of a generation.
ATTACK_SIZE	5	Number of attack sequence in a boss.
SIZE_OF_ACTION_IN_ATTACK	7	Number of actions in an attack sequence.
survive_rate	0.5	Proportion of population that will survive in each selection.
mutation_rate	0.04	Rate of each action to be mutated.
expect_timeUsed_toBeat	120	
expect_attackUptimePercentages	0.4	
expect_playerHP_left	80	
weight_timeUsed	4	
weight_attackUptimePercent	5	
weight_playerHP_left	1	
weight_isBeaten	3	
acceptable_avg_diff	0.0005	

Table 7: First run configuration.

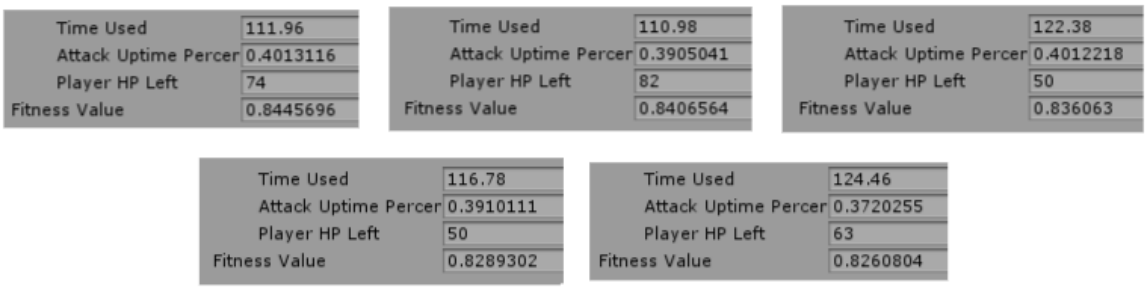
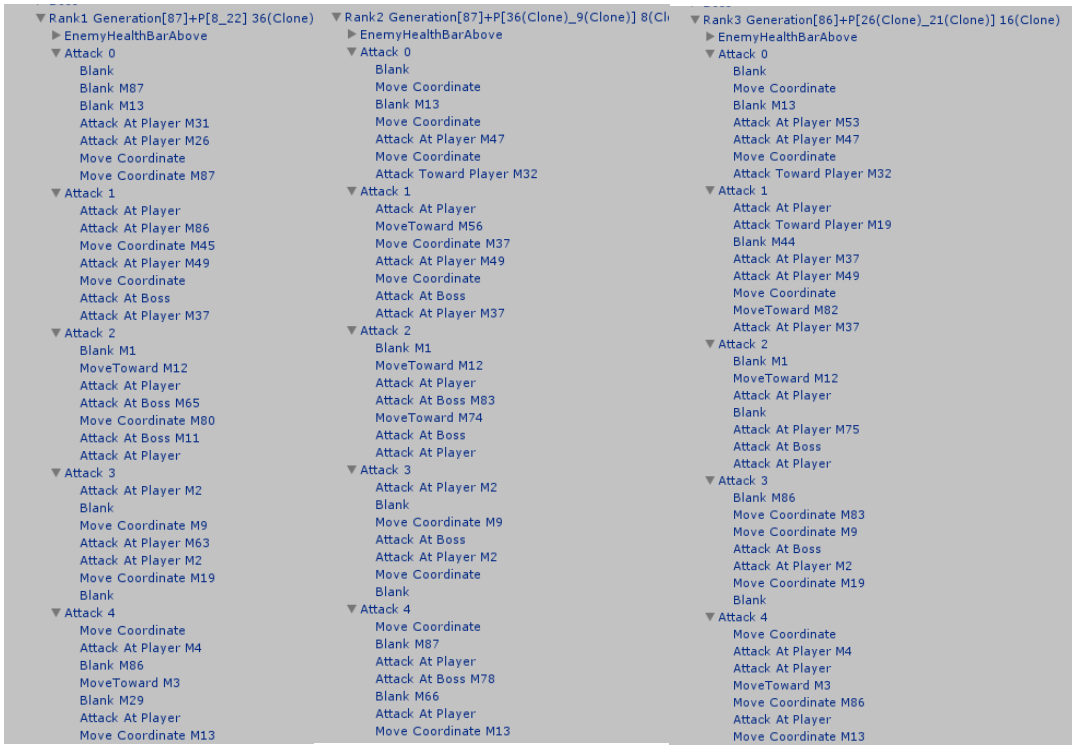


Figure 10. Fitness values of 5 bosses with highest fitness values.

The average fitness value converged at 89th generation with 0.73438 average fitness value (the previous generation had 0.73452 average fitness value). Average fitness values swung around 0.72-0.75 before it converged. Fitness values of five bosses with the highest fitness values are shown in figure 10 and their actions are shown in figure 11. All expected parameters were close to the intended value, except the remaining HP, with the lower fitness bosses being quite different. The higher fitness bosses did not have this problem.



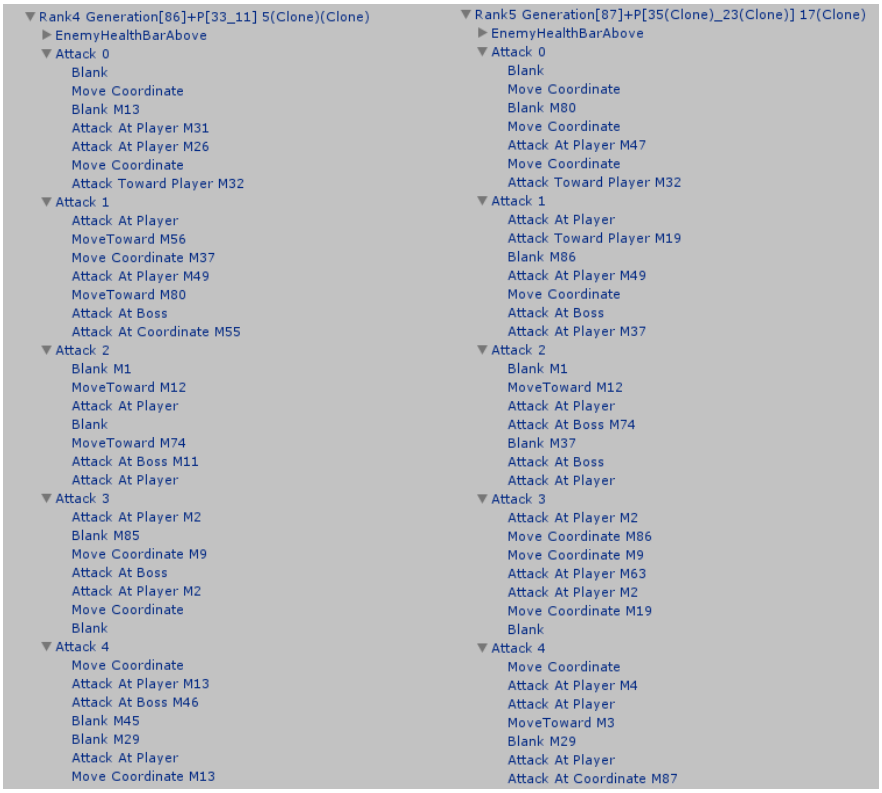


Figure 11. Top 5 fittest bosses from first run and their actions.

Inspections from five bosses with highest fitness values are as below:

- Most attack sequences had at least 2 movement action.
- Move to a coordinate actions' coordinates were at a border or a corner of arena.
- Move toward player action appeared less than move to a coordinate.
- Move toward player actions' distance offset size were more than 5.
- The attack at coordinate action did not appear in these bosses.

The second run configuration, with reduced size of actions in an attack sequence, increased mutation rates and lower expected player health left after battle, is shown in the table below:

Name	Value	Description
Number of game instances	80	Number of game instances and total population of a generation.
ATTACK_SIZE	5	Number of attack sequence in a boss.
SIZE_OF_ACTION_IN_ATTACK	4	Number of actions in an attack sequence.
survive_rate	0.5	Proportion of population that will survive in each selection.

mutation_rate	0.06	Rate of each action to be mutated.
expect_timeUsed_toBeat	120	
expect_attackUptimePercentages	0.4	
expect_playerHP_left	30	
weight_timeUsed	4	
weight_attackUptimePercent	5	
weight_playerHP_left	1	
weight_isBeaten	3	
acceptable_avg_diff	0.0005	

Table 8: Second run configuration.

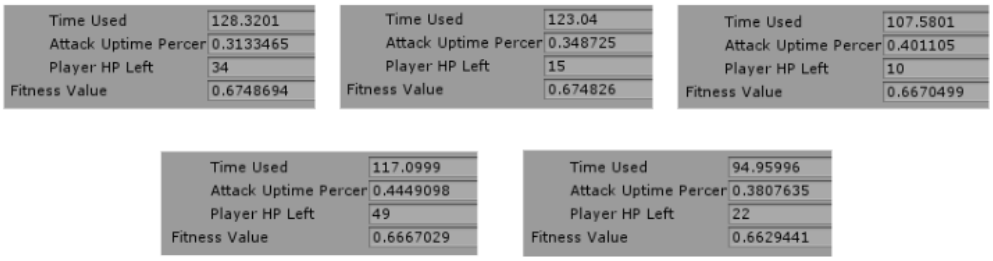


Figure 12. Fitness values of 5 bosses with highest fitness values of second configuration.

It converged at 70th generation with 0.5538 average fitness value (previous generation had 0.5536 average fitness value). Average fitness values swung around 0.54-0.56 before it converged. This configuration fitness values of five bosses with highest fitness values are shown in figure 12 and their actions are shown in figure 13. Most expected parameters were close to the intended value.

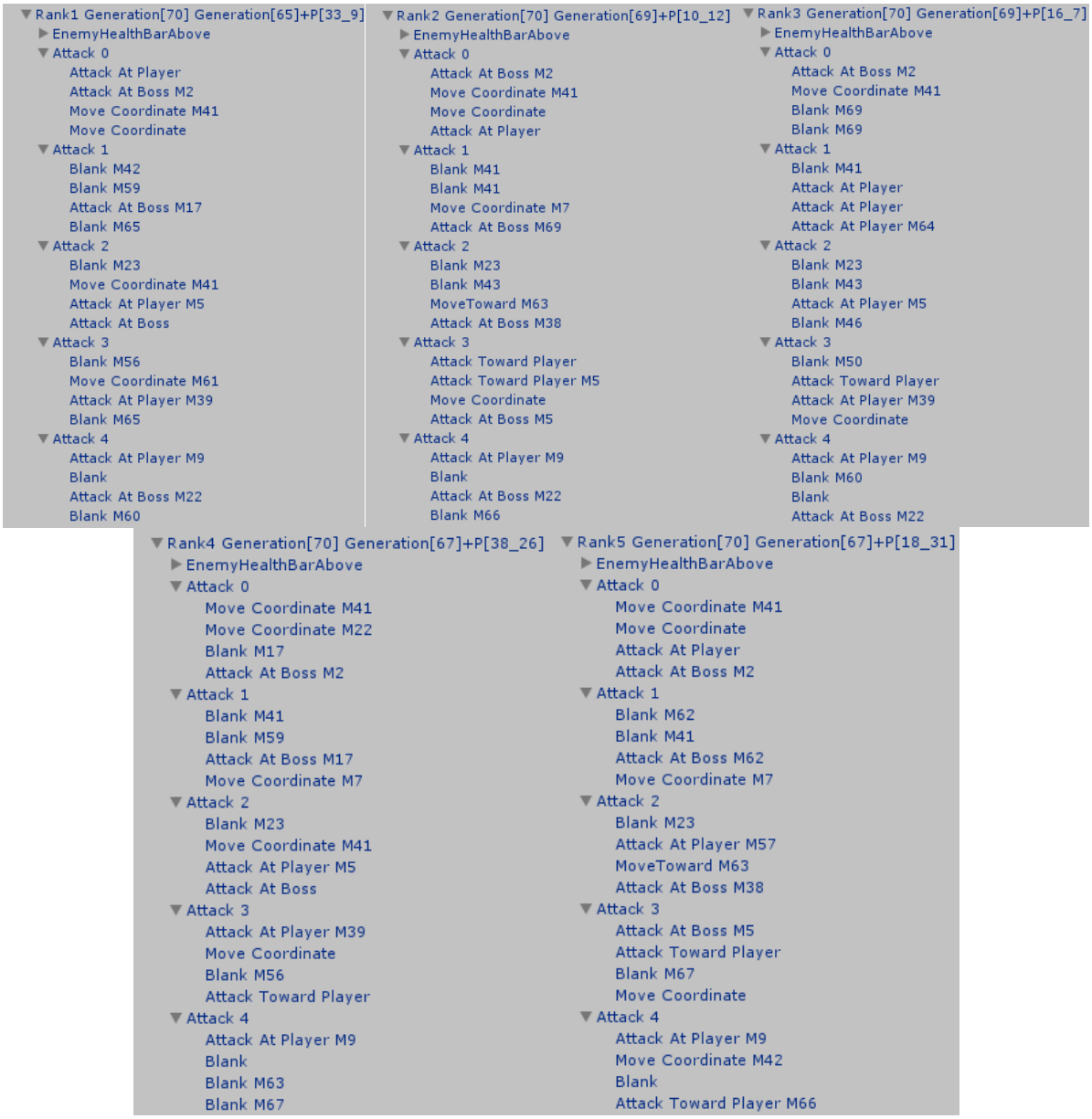


Figure 13. Top 5 fittest bosses from second configuration and their actions.

Inspections from the five bosses with the highest fitness values of this configuration are as below:

- Blank actions appeared more than the first configuration and largely came from mutations. This implied that blank actions replaced weak action so that strong actions would have more chance to perform.
- Non-blank Actions frequently appeared in the same slot for these bosses could be considered as strong actions.
- Movement actions tended to move away from the player as in first configuration.
- There was no attack at coordinate action.

From both runs, movement actions made the boss harder to be attacked and resulted in more time to beat the boss. For attack actions, they contributed to both attack uptime value and expected time to beat the boss because the player must dodge attacks. Weak actions such as attack at coordinates (uncertain to hit) or actions with bad parameters were replaced by blank actions or stronger actions.

4.2 Survey Results

Five bosses with highest fitness value from second configuration were put in to a demo game for testing with a survey form that had question about the generated bosses. There were 6 people who took the survey, consisting of 3 who had seen the game before and 3 who had not seen the game before.

Feedbacks:

- Bosses were playable and in acceptable difficulty level.
- Some bosses were clearly more difficult than others.
- Boss difference from each other was low.
- Low variedness of boss attacks.
- Attack sequences were hard to notice and distinguish.

5. Conclusion and Discussion

5.1 Conclusion

This boss generation method could generate appropriate bosses that could be defeated and taken reasonable amount of time to be defeated with expected performance measured by fitness values but due to low pool of possible action class, difference of bosses was low.

5.2 Encountered Problems and Further Improvement

5.2.1 Encountered Problems

- Lack of knowledge and experience in genetic algorithm. Other function or operators could be used for survival of a generation, crossover, mutation and convergence criteria which could result in a better outcome.
- Spent too much time on ML-Agents reinforcement learning player agent AI.
- Low pool of action class might be the reason for low variedness of generated boss genes.
- Sequences were randomly chosen, causing patterns of bosses in each fight to vary, which resulted in requirement of better convergence criteria and having to average each boss's past battle data.

5.2.2 Further Improvement

This boss generation technique is highly coupling with the player because it generates bosses from player AI battle data. As in many games, players learn an attack sequence by acknowledging and distinguishing the sequence from an indicator such as the attack sequence name or animations. Player learning about attack sequence is an interesting point for actual boss in many games, but to do that, player AI with the ability to learn attack sequences and more fitness functions are required.

References

[1] **Tiga Genre Report**, https://tiga.org/wp-content/uploads/2016/03/4257-TIGA-Genre-Report-2016_v3.pdf, last edited on 5 July 2016. Accessed 15 November 2019.

[2] **List of video game genres,** https://en.wikipedia.org/wiki/List_of_video_game_genres, last edited on 7 November 2019. Accessed 15 November 2019.

[3] Christina Gough. **Most popular video game genres 2018** https://www.statista.com/statistics/189592/breakdown-of-us-video-game-sales-2009-by-genre/, last edited on 3 July 2019. Accessed 15 November 2019.

[4] **Action game – Wikipedia**, https://en.wikipedia.org/wiki/Action_game, last edited on 23 October 2019. Accessed 15 November 2019.

[5] Ernest Adams, Andrew Rollings, **Fundamentals of Game Design (2007)**.

[6] Melanie Mitchell, **An introduction to genetic algorithms**, MIT Press, Cambridge, MA, 1996.

[7] Vijini Mallawaarachchi, **Introduction to Genetic Algorithms — Including Example Code**, https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3, 8 July 2017, Accessed 15 November 2019.

[8] **Genetic algorithm – Wikipedia**, https://en.wikipedia.org/wiki/Genetic_algorithm, last edited on 16 October 2019, Accessed 15 November 2019.

[9] Pichit Promsutipong. **Automatic enemy behavior pattern generation for 2D avatar-based action game using data mining technique.** Chulalongkorn University, Academic Year 2016.

[10] Theodore Agriogianis, "**The Roles, Mechanics, and Evolution of Boss Battles in Video Games**" (2018). Undergraduate Honors College Theses 2016-. 45. https://digitalcommons.liu.edu/post_honors_theses/45

[11] Linkmstr, **Dragalia Lost - High Midgardsormr's Trial: Expert (Valentine's Hildegarde POV) [Deathless]**, https://www.youtube.com/watch?v=tFRrWVSkAXE, Published 14 October 2019, Accessed 6 May 2020.

[12] Mekkah Dee, **FFXIV OST Eden Boss Theme #1 (Force Your Way) SPOILERS,** https://www.youtube.com/watch?v=6Pd0ALPOhnQ, Accessed 15 November 2019.