AI for Dental Defenders

Mohamed Albashir, Kinshuk Basu Chowdhary, Ujan Sengupta
North Carolina State University

Abstract

Dental Defenders is a 2D (top-down view) survival game in which the player must protect itself and a tooth from incoming enemies. The tooth is stationary in the middle of the screen and the player needs to kill the enemies in order to ensure their collective survival. Points are gained by combination of eliminating enemies and the total time that both the player and the tooth manage to survive. The objective of the game is to maximize the points earned. If either the player or the tooth runs out of life, the game ends.

Our objective is to create a game like Dental Defenders where we want to try out, invent and implement novel enemy behaviors to make the survival game interesting. The variety of enemies will range from simple seekers with a one-track objective of reaching the tooth, to enemy bosses that can return fire and pursue the player. Additionally, we plan to add a helper bot for the player that can function as a stationary turret to fire at nearby enemies. The goal of the project is to create various interesting types of enemy bots demonstrating different AI techniques to defeat the player, and creating an AI to control the player and garner maximum points.

Introduction

This project involves movement and decision making at the core of its AI. 'Dental Defenders' is a game in which there exists a central objective (the tooth), which the player needs to defend against incoming enemy characters.
The player has the ability to move in all directions, as well as shoot bullets. If a player attacks an enemy, it temporarily abandons its pursuit of the objective and pursues the player. The player then has to either kill the enemy, or get a certain distance away from it to cancel its pursuit. If an enemy comes in contact with the player, the player takes some damage.
The game is over if either the player dies, or a certain number of enemies reach the tooth. As there might be multiple enemies on the screen at the same time, the player needs to prioritize which enemies it will focus on, and in what order.

Background

Character and enemy movement is done using basic movement algorithms, combined with obstacle avoidance and path finding.
The environment consists of open spaces, as well as walls. It is represented in the form of a dense tile graph, removing the need for intensive obstacle avoidance implementations. The player as well as the enemies find their way around the environment using a path-finding algorithm, and then following the resulting path. Since the player and some enemy units can shoot bullets, this behaviour would be accomplished using a simplified version of a pursuit algorithm.
The most essential, as well as intricate part of the AI for this game would be the decision making system. Since each decision would have multiple factors affecting it, the parameters used for decision making would need to be carefully calibrated.

Environment

The layout for the game is rather simple. The tooth is formed in the middle of the screen, and does not move. This tooth is the objective that the player needs to protect while all enemies try to approach it. The map is bounded in the visible area, and the movement space is littered with randomly placed small obstacles and small walls. Both the player and the enemy units are required to maneuver around these obstacles. At the beginning of the game, the player is spawned at a constant location (E.g. slightly above the objective), and is then free to move around. Enemies are spawned randomly, according to rules which are discussed in later parts. The entire map is stored in the memory in the form of a directed graph, which is a data structure formed with connecting different parts of it called nodes using edges. Each edge will have a specific weight, that represents the cost of traversing that edge to reach the other node, and a direction. It's also singly connected.

Graph

The entire screen is represented in the form of a grid, with each tile a node in the game graph. Nodes are connected with the 8 surrounding nodes using 2 kinds of edges, side edges and diagonal edges. Side edges cost 1 unit, and diagonal ones cost 2 units. Tiles that are located where an obstacle is, are considered invalid nodes, and are hence not connected with their neighboring nodes, or in other words, theyre not considered nodes.

Keeping that in mind we generate the graph with its nodes and edges. We also keep track of invalid nodes for future references.

The movement of characters in Dental Defenders is not limited to using just the graph nodes, as that would require a high amount of computation - an A* search every draw cycle and would limit character motion to only 8 directions. We use regular steering behaviours like Seek, Evade and Pursue to traverse the environment and utilize graph nodes for movement only while doing path-following.

Obstacles

Since obstacles are considered invalid nodes, the different game characters wont ever find themselves in a path going through them. They are positioned each at a quadrant of the screen, at the same distance from the edges. Obstacles dont deflect projectiles; they simply block them. Because thats the case, they affect the behavior of characters, since they require a line of sight to be able to shoot at their target. Because this is a 2D games, obstacles are only distinguishable from other parts of the map by color. They're filled with a color that's also different from the one that fills that tooth.

Shortest Path

Enemies spawn and have a different behavior, but eventually the main objective is to attack the tooth. Should an enemy choose to move towards the tooth, or towards the character, it will try to achieve an optimal path towards the target. We have used A* algorithm to compute a path from the location of the character to the target decided by the characters decision making mechanism.
Since the path-finding algorithm will run quite often during the course of the game, we would like to minimize its runtime, while still obtaining an optimal path. In order to do this, we need to use a heuristic for A* pathfinding that would minimize the fill. We could use a heuristic such as Euclidean distance, which would give a really small fill, and also return a shortest path every time in our environment.
Sometimes, from a game experience point of view, it is more interesting to the player if some AI units take non-optimal paths, as it adds a hint of unpredictability. This can be implemented by using a greedy, best-first approach in A*, where the next node is chosen based almost solely on the value of the heuristic. Depending on the layout of the environment and the resulting graph, this approach will still often yield the shortest path, but may cause the AI to take a different, slightly longer path. Some enemies could use this approach to path-finding.

Game Objects

Each element that exists in the game environment is treated as a game object. In this section, we describe the functionality as well as implementation of these objects.

Player The player, or the player character is the central character of the game. Its function is to defend both itself and the tooth from enemy agents. Since it is meant to emulate as closely as possible the behaviour of a human playing the game, its AI will be the most complex one in this game. The player has a health pool of a 100 points, has no fixed movement patterns, and can fire bullets which damage enemies. Below, we have detailed player response to certain situations that would arise during gameplay -

- If there is no immediate threat to either the tooth or the player, the player positions itself in a defensible position around the tooth. In our implementation, the player patrols around the tooth as long as there are no enemies on screen that warrant its attention.
- Not all enemies on screen are considered as immediate targets for the enemies. Due to the presence of obstacles around the tooth, it is a much safer strategy to limit the player's position to inside the area. Since there is a constant onslaught on the tooth's position, moving the player to the outer parts of the map would limit its ability to counter enemies closing in on the tooth. Therefore, in most cases, the player only considers enemies that have come within the inner area as threats. This , however, does not apply to the Enamelator, as it can attack the tooth from a long range. Thus, the player moves to attack Enamelator units the moment they appear on screen.
- Once the player starts attacking an enemy, it will continue this attack until either the enemy is dead, or there is an immediate threat to the tooth or the player.
- If an enemy starts pursuing a player, it will shoot at the enemy. If the enemy dies on its way to the player, it will resume normal operations. Else, if the enemy gets within a certain radius of the player, the player will then begin to evade the enemy until it is safe again.
- Certain enemies can also fire bullets at the player. In this case, we need to find a balance between evading bullets fired at the player, and firing at the enemy. In a case like this, the player can avoid bullets by moving slightly to the side, ensuring it does not break line of sight with the enemy, and then continue firing. This feature remains to be added.
- The game is considered to be over if either the player dies, or the tooth takes enough damage. Therefore, the player has a responsibility to protect both itself and the tooth. In cases where there is a threat to only one of them, the course of action is obvious, i.e. protect the one under attack. If both of them are under imminent attack, the player can prioritise its attention based on the distance of the enemies. For example, the player could be protecting itself while it is under fire, when it suddenly notices that an enemy has gone past a critical radius around the tooth. At this point, it must focus its attention on going to the tooth's rescue.
- In addition to the above strategy, there is another situation in which the health points of either the player or the tooth have fallen below a certain threshold. In such a case, the player must always prioritise the defense of the injured asset. If this is not done, the game is likely to come to an early end.

When it comes to engaging enemies, the player needs to position itself in order to get line-of-sight (LOS) onto the enemy, so that it can attack. Line of sight is established by implementing a ray cast from the player's position to the enemy's. If the ray does not encounter any obstacles on the way, LOS has been established, and the player can proceed to attack the enemy.

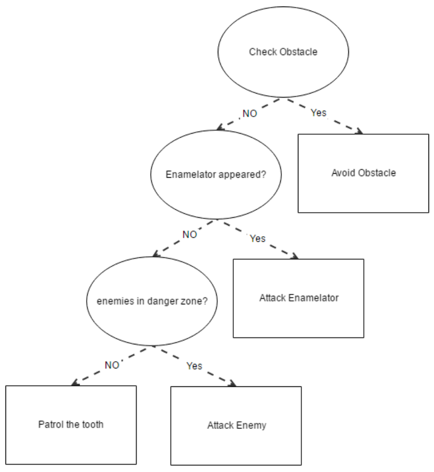


Figure 1: Basic decision tree to represent enemy tracking by Player AI

Another issue that the player AI needs to deal with is evading enemies. We are currently using a graph based approach to this. When a player tries to avoid an enemy, it chooses a node in the opposite direction, and pathfinds its way to that location. While this may seem computationally expensive at first glance, that is not necessarily the case. Since the evasion will take place to relatively nearby nodes, the fill of the pathfinding algorithm will be very low. Also, it saves us the computational cost of continuous ray casting. When evading multiple enemies, a blending approach is taken. Multiple escape points would be chosen, one for each enemy, Since these points are essentially position vectors with respect to the character, they can be added and scaled to create a new escape point. This resulting point is then checked for validity, i.e. ensuring it does not lie on an obstacle, does not take the player directly to an enemy. If these checks are passed, the player will move to said point. As the player moves, it is still continuously scanning its surrounding for enemies, and can change its course on the fly if new threats present themselves.

The complete implementation of the above is not included is this project.

Enemies There are 4 different kinds of enemies in the game. There can be multiple instances of each type of enemy on the screen at one time, except for the Enamelator, which functions as a "boss enemy", and will therefore have a maximum of one instance at a given time. Each instance of a given enemy type will have the same characteristics and behaviour. The different types of enemies are explained in detail below-

- **Lactus** - This is the most basic of all enemy units. It has 20 health points, making it the easiest enemy for the player to kill. Once it spawns at the edges of the playing area, it starts to wander across the screen. This behaviour continues until either the player or the tooth enters within the perception radius of the unit. In such a situation, the unit will then cease to wander and actively target the player or the tooth. If the player moves out of the perception radius, the enemy unit will then begin to wander again.
  If a situation arises in which both the payer and the tooth are within the perception radius, Lactus units will prioritize the tooth.

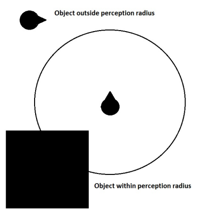

Object outside perception radius
Object within perception radius

Figure 2: Simple illustration of the perception radius

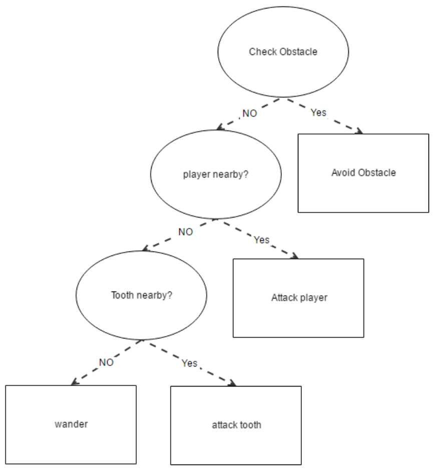


Figure 3: Basic decision tree to represent the Lactus Behavior

- **Fructus** - Enemy Fructus units aim directly for the tooth. They have 40 health points per unit, making them slightly harder for the player to kill. Once Fructus units spawn at the edges of the screen, they use Seek to move towards the tooth. If the player appears within it's perception radius, it temporarily gives up on its pursuit of the tooth and moves towards the player instead. Fructus units do

not have any ranged attacks or bullets to inflict damage with. Therefore, the only option for them to inflict damage is to come in contact with the player character.
If the player moves out of range, the unit will direct its attention back to the tooth. However, every time the player attacks a Fructus, it will pursue the player for at least a set amount of time.

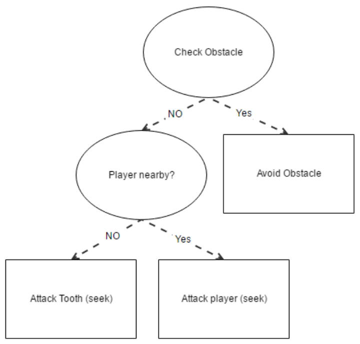


Figure 4: Basic decision tree to represent the Fructus Behavior

- **Streptus** - These enemies are the second strongest class of enemies in the game, with 60 health points per unit. Streptus units also have guns with which they can attack the player. Once they spawn in the game environment, they use Seek to move towards the tooth. However, if the player attacks a Streptus unit, it will stop and try to shoot the player. However, there's a chance that the player will not be in Streptus' line of sight. Therefore, we will use ray casting to establish if line of sight exists. In most cases, depending on the density of obstacles, line of sight will exist, as the player will just have fired on the Streptus. In case this is not true, the Streptus will still fire in he direction of the player for some time, and then aim for the tooth again.

- **Enamelator** - This enemy acts as a boss enemy in the game, and spawns very infrequently. It has 80 health points, and can attack the player using guns, as well as a homing rocket. This class of enemy will actively target the player and constantly pursue it, in an effort to pressure the player. The homing rocket is used at certain intervals by the Enamelator, and only homes to the location of the player at the time of launch. This feature can be implemented using pathfinding. Normal attacks are carried out using bullets.
If the player stays out of sight of the unit for a long time, the Enamelator will change its focus and moves towards the tooth. This would force the player to come out of hiding and engage the enemy, at which point the Enamelator can focus on the player again.

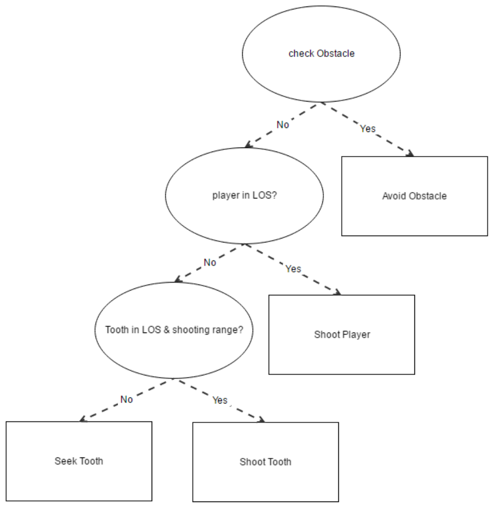


Figure 5: Basic decision tree to represent the Streptus Behavior

Bullets Bullets are a class of game objects which are used by both the player as well as the enemies. They originate at the same location as the character shooting them, and travel in a straight line thereafter, using kinematic seek behaviour. If a bullet hits a character, it deals 10 points worth of damage. Else, bullets will disappear if they hit either an obstacle, or the edges of the playing area.

Difficulty Adjustment

Usually game developers refine the gaming experience based on feedback they get from testing the game. They modify the game until they feel its balanced and enjoyable. While we do not propose changing the structure of the game, we decided to make the difficulty of the game change as the game progress. This is similar in concept to the rubber band effect in racing games, where the distance between the leading car and the trailing one doesn't exceed a certain maximum value. That way, there is always a chance for the trailing car to win the race. Here we try to achieve a similar goal by varying the number of bullets required to defeat the enemies or make the player take less damage from different attacks. The decision is made based on both the characters rate of health points drop as the time progress.

However, this change in difficulty will not happen all at once. A sudden extreme change in difficulty will not be a desired behavior, because it will also create an imbalance. We choose to avoid that by checking the rate drop every Δ T. First there will be a predefined rate which the player's HP is expected to decrease at. Should the actual rate be less than the expected one beyond an acceptable radius, we implement the change in difficulty and periodically check to see if the decrease in the players/tooths HP maintains the same (or a similar) rate, if the rate is not changed after the difficulty adjustment is implemented, we keep the difficulty at the same level, because the number of enemies will increase and its expected the game wont remain the same in terms of ease of play. If the rate changes back to one close to the one we set at the beginning, we dont change the settings to make it easier, unless it changes beyond a certain radius. Should that happen, we adjust the difficulty to make it easier for the player to kill enemies and make him sustain less damage from bullets and collisions.

The player also has a rampage mode. It gets into it when its HP drops below 25% of its maximum health. Visually, the players color and appearance changes when it gets into this mode and it sustains less damage from bullets and other collisions from enemies. This mode also makes the player's bullets more powerful. The motivation behind this mode, is to make the game more interesting as it starts to come to an end. It also derives from the sense that a soldier who has minimal life remaining, will have nothing to lose, and will inflict more damage to the enemy.

Evaluation Methods

This section is concerned with the empirical evaluation of all our game characters in terms of their behaviour, survival time, damage caused, etc. Naturally, since the player and the enemies have different objectives and multiple ways to accomplish those objectives, the same evaluations method won't apply to all the characters.

To maintain consistency in our results, the results from each test would be performed ten times and the average results would then be compiled and presented. We presume that the actual in-game behaviour of the individual characters would not differ greatly from the test evaluation results, since the the presence of multiple characters would only add a permissible degree of random noise to the game.

Lactus

Although the Lactus lies in the lowest rung of enemies, it's behaviour is governed by a number of causal and correlated factors. The Lactus wanders around till it encounters either the player or the tooth in it's perception radius, so we shall test the average time it takes for the first encounter to occur. If the first-encounter time is significantly large, it would mean that the unit is ineffective in causing damage to the player or the tooth.

The first test scenario involves 1 lactus and the player, who is tasked with escaping the enemy and subsequently killing it. Since the lactus does not wield firepower, the only way it causes damage to the player is by colliding with it. We shall test the time that a single lactus survives in the game and the amount of time it spends wandering vs. the amount of time it spends in pursuit of the player.

The second test scenario involves a group of 5 lactuses and the player. Since the lactus is a basic enemy, it has a higher spawn rate than the other germs. It stands to reason that at a given point in time during the game, there would be multiple lactuses wandering about (even though there may not be other enemies). We test the same metrics as in the case of a single lactus but by increasing the number of enemy units, we wish to offset the fact that the game map is large and the first-encounter time might be significant for a single unit.

Fructus

This is the first of the three types of enemy that actively seek the tooth in order to cause maximum damage to it. If the player shoots at a fructus, it will abandon pursuit of the tooth and shift its target to the player. Since it does not wield weapons, like the lactus, it can only cause damage by physically colliding with either the tooth or the enemy.

We shall test the efficacy of a fructus to cause damage to the player and the tooth by measuring the amount of time that a single unit survives in the game. Thus, the test scenario would have 1 enemy unit and the player. Since the amount of damage it can cause is constant, we check if a single unit is able to reach the tooth before it gets killed by the player.

The fructus needs to be hit by player fire in order to change its target of pursuit. However, it gives up pursuit of the player to continue seeking the tooth as soon as the player goes out of its perception radius. In a situation where the player shoots at the fructus from a distance greater than its perception radius, the basic enemy logic would imply that the unit continues towards the tooth while disregarding the player (since it is already outside the radius of perception). Thus, we further test two things - 1) various perception radii of the fructus so that it can make an intelligent decision about what target to pursue when, and 2) at what combination of health level and distance from tooth, it should disregard all player fire and continue towards the tooth.

Streptus

The Streptus is the first of two enemies that wield weapons and can fire at the player. As such, it has the potential to cause variable damage to the player and the tooth, depending on how much fire it can release before it is killed or it reaches the tooth. When a streptus is fired at, it will stop, shoot at the player for at least 2 seconds or till the line-of-sight is blocked by an obstacle, and then continue towards the tooth.

The test scenario will consist of 1 enemy unit and the player. We shall evaluate how much time the enemy is able to survive in the game and how much damage it causes to the player and tooth, averaging over 10 runs of the test. It could be the case that the player shoots at the streptus and immediately hides behind an obstacle. In this case, though the line-of-sight is not established, we shall enable the streptus to spray fire in the approximate direction for at least 2 seconds, so as to prevent an immediate follow-up attack by the player. We shall be testing various arbitrary timings (in addition to the 2 seconds limit) to evaluate what time period enables maximum survival in the game.

Enamelator

The Enamelator is the strongest class of enemies and has multiple ways to cause damage to the player. It wields a gun to shoot bullets as well as homing rockets, which can use path finding to reach the player's last known location. Since this enemy is powerful and constantly seeks the player (unless it is out of sight for a stipulated amount of time), it will be spawned infrequently.

The test scenario will contain 1 enamelator unit and a player. We shall test the amount of time a single enamelator survives in the game, which is also the amount of time required by the player to actively kill the enamelator. Since this enemy can cause damage in multiple ways, we shall evaluate and differentiate the damage cause by the bullets vs the homing rockets. Also, since it benefits the player to remain out of sight in order to escape the enemy attacks but it's harmful for the tooth to remain without protection from a boss enemy, we shall evaluate the average time the player spends hiding from the enemy vs the time that it spends actively engaged in battle.

Since the homing rockets are an interesting form of weaponry, we shall be testing the efficiency of the rockets in reaching the player and causing damage to it. After a rocket reaches the player's last known position before it was fired, it will expect to collide with the player. If the player has moved by then and the rocket does not face collision, it will continue along a straight line path from that point till it hits an obstacle or the screen boundary (and then explode). We shall evaluate the number of rockets fired by the enemy and compare it to the number of rockets that actually hit the player, thus giving us a fairly useful efficiency metric.

Player

We shall be analyzing the performance of the players AI in the presence of various enemies to evaluate its performance. Since the player has a dual objective of self-preservation as well as protecting the tooth, we need metrics that would compare and contrast the effectiveness of the player in doing both those things.

The tests would be conducted on a regular game-play scenario, with enemies spawning at different intervals and with varying frequencies. We shall evaluate the amount of time that the player survives in the game, the total number of enemies eliminated and the final health of the player vs. the tooth. Ideally, the dental "defender" would sacrifice itself to protect the tooth, which implies that more often than not, the game should end with the player getting killed rather than the tooth getting destroyed. We shall be testing this hypothesis by evaluating the health of the tooth vs. player health for all our test runs.

We also aim to evaluate the amount of time the player is idle compared to the time it spends engaged in combat (shooting, fleeing enemies, hiding, etc.). Depending on the time it spends doing each behaviour, we can effectively predict the performance of the player upon the increase or decrease in difficulty level. Although we haven't implemented difficult adjustment yet, we shall run our tests for different difficulty levels before reporting the final evaluations.

Results

Enemy Evaluation

In this section, we evaluate the effectiveness of each type of enemy against the player, in a one-on-one situation. One instance of the enemy is spawned at a time, and certain metrics are measured at the time of its death. These metrics are then averaged out over multiple runs.

Enemy	Avg Time to Die	Avg Damage to Player	Avg Damage to Tooth
Lactus	58.2s	0	0
Fructus	13.6	9	0
Streptus	13.5	24.5	2
Enamelator	17.2	28	1

Table 1: Performance of Enemy units against Player AI

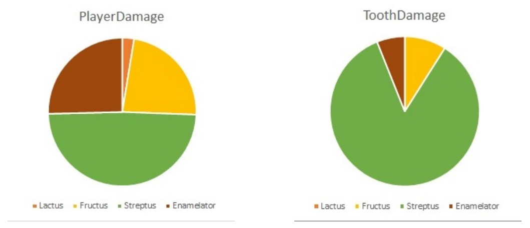
- **Lactus** - Lactus is the weakest enemy unit in the game. Since its movement is dictated by wander only, its interactions with either the tooth or the player are only a matter of chance. As a result, its survival time is disproportionately long as compared to the other types. However, we can see that a sole Lactus unit poses virtually no threat to the player or the tooth, since it can be shot down extremely quickly once it gets close to either, without a chance to inflict any damage whatsoever.

- **Fructus** - These units are programmed to seek the tooth, or to attack the player if it comes within their perception radius. As a result, they encounter the player fairly quickly, and therefore have much shorter lifetimes. However, as a basic unit, they have no guns to fire and need to come in contact with either the tooth or the player in order to inflict damage. Thus, the player is usually able to kill Fructus units without suffering much damage.
- **Streptus** - Streptus units also seek the tooth, or the player if it is within their perception radius. Thus, they too have a short lifetime, as the player quickly detects them and moves to kill. However, Streptus units do possess guns, which they can use to inflict considerable damage to the player and the tooth.
- **Enamelator** - As a 'boss' enemy, with a higher health pool, Enamelator units last somewhat longer against the player, and also cause more damage on average. However, due to their slower rate of fire, the damage they inflict is not significantly more than that of Streptus units.

Comparative Evaluation

In this section, we compare the performance of different player AI strategies, with and without difficulty adjustment in a normal run of the game, and compare the collected statistics.
As mentioned above, the general strategy for the AI is to ignore enemy units on the screen until they get within a certain radius. At this point, the player AI prioritizes an enemy, and moves to attack.

AI focuses on Enamelator, without difficulty adjustment
In this approach, while we maintain the above approach to attacking enemies, with one modification. Anytime an Enamelator appears on screen, we prioritize it as the target and move to attack it. This approach is motivated by the fact that it is the enemy with the maximum potential for damage, and should be taken care of quickly.



(a) Damage to Tooth (b) Damage to Player

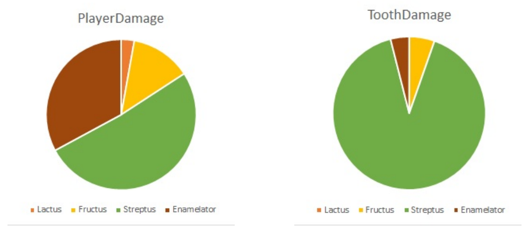
Figure 6: Contribution to damage by Enamelator(Brown), Streptus(Green), Fructus(Yellow) and Lactus(Orange

This graph shows the contribution to the total damage done per run, by each type of enemy. As we can see in the graphs, Streptus units do, by far, the most damage to both the tooth and the player.

- Avg Duration - 36.9 seconds
- Avg Enemies killed - 5.5
- Avg Player health - 0.5
- Avg Tooth health - 96.0

AI focuses on Enamelator, with difficulty adjustment
The AI's approach to defeating enemies remains the same as in the previous section. However, we have now enabled the difficulty adjustment mechanic in our code. Since the player takes damage quite soon under the onslaught of enemies, we would expect this change to increase the lifetime of the player.

- Avg Duration - 44.4 seconds
- Avg Enemies killed - 5.7
- Avg Player health - 2.0
- Avg Tooth health - 86.5



(a) Damage to Tooth (b) Damage to Player

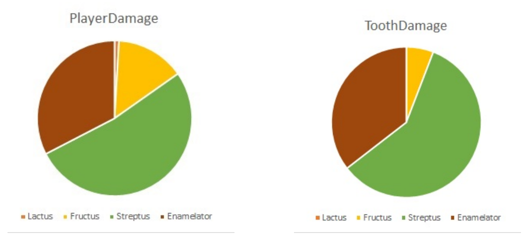
Figure 7: Contribution to damage by Enamelator(Brown), Streptus(Green), Fructus(Yellow) and Lactus(Orange

Analysis
In both the above methods, we can see that Streptus units inflict a majority of the damage onto the player and the tooth. This calls into question our methodology of immediately targeting the Enamelator. Since Streptus units are the biggest damage dealers, it might be more interesting to focus on them instead.

AI focuses on Streptus, without difficulty adjustment

- Avg Duration - 38 seconds
- Avg Enemies killed - 4.4
- Avg Player health - 4.5
- Avg Tooth health - 119.5

In this case, we choose to give a priority to Streptus units that get within a dangerous distance to the tooth. Since they can inflict damage by firing bullets, their potential for damage is quite high, and we seek to limit that damage.

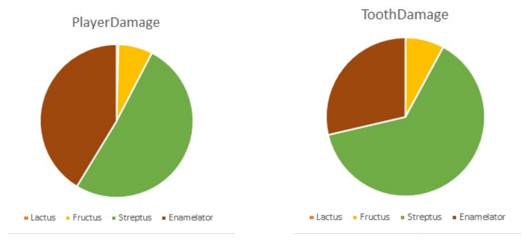


(a) Damage to Tooth (b) Damage to Player

Figure 8: Contribution to damage by Enamelator(Brown), Streptus(Green), Fructus(Yellow) and Lactus(Orange

AI focuses on Streptus, with difficulty adjustment
The AI's approach to defeating enemies remains the same as in the previous section. However, we have now enabled the difficulty adjustment mechanic in our code. In this case too, we expect the player's lifetime to increase.

- Avg Duration - 53 seconds
- Avg Enemies killed - 5.5
- Avg Player health - -1.0
- Avg Tooth health - 161.5



(a) Damage to Tooth (b) Damage to Player

Figure 9: Contribution to damage by Enamelator(Brown), Streptus(Green), Fructus(Yellow) and Lactus(Orange

Analysis
As expected, we see a more even balance in the graphs depicting sources of damage. Once the player AI starts focusing on Streptus units, the contribution to total damage from the Enamelator increases. Further, we can see that this approach to the player AI is a step in the right direction, as evidenced by the increase in average player lifetime.

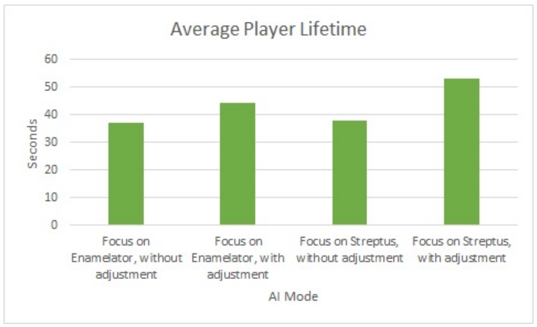


Figure 10: Average lifetimes for the player, in different scenarios

As a trend, including the difficulty adjustment mechanic increases the expected lifetime of the player. The best results are seen when the player chooses to focus Streptus units, and is helped by dynamic difficulty adjustment.

Conclusion

Dental Defenders doesn't require a lot complex physics or convoluted logic to engineer. However, it's not an easy task to implement an autonomous AI that plays the game and controls the enemies' movement and decisions intelligently. To that effect, we create and evaluate multiple AI agents in order present a game with a robust empirical test background and a visually appealing experience. From running multiple simulations of the game with different parameters, we can draw the following conclusions-

- Performance of enemies : The different enemies are equipped with different kinds of behaviour and there is no single characteristic that governs the enemies (other than the fact that they want to destroy the tooth). As a result, it is interesting to see how the various enemy AIs interact with each other. Although we haven't specifically used a strategy pattern amongst the enemies, the evaluation results from the test runs ought to display a pattern of enemy behaviour that would be interesting to analyze.
- Player performance : The difficulty of the game is solely dependent on the performance of the player. As the player gets more adept at destroying the enemies and protecting the tooth, the frequency of enemy spawning increases and more high-level enemies are gradually spawned. The evaluation of the difficulty level increase/decrease would give us an idea about how well our decision trees can be trained on the data that is provided. It would also tell us if there is a pattern to the learning curve of the AI.
- Project Structure : As is the case with any large scale programming effort, the project structure and code organization is vital to a successful execution. This issue is further compounded when multiple people are working on the same code. As a result, we have developed a structure that is intuitive and easy to understand, even for someone going through our code for the first time. Carefully organized package structures, adept use of OOP principles like abstraction and inheritance and a proper class hierarchies have been implemented in order to make debugging and updating easier.

References

- Dental Defenders Game -https://cas002.itch.io/dental-defenders-saga-of-the-candy-horde
- Game AI Reference - https://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1
- Artificial Intelligence for Games, 2nd Edition - Ian Millington, John Funge
- The Case for Dynamic Difficulty Adjustment in Games, Robin Hunicke, Northwestern University
- Improving player balancing in racing games, Jared E. Cechanowicz, Carl Gutwin, Scott Bateman, Regan Mandryk, Ian Stavness