**FACULTY OF SCIENCE, ENGINEERING AND COMPUTING**

**School of Computer Science and Mathematics**

MSc DEGREE IN Games Programming

CI7800 Digital Media Final Project

DESIGN REPORT

Name: Hamza Oudrhiri

ID Number: K1910115

Project Title: Pokemon Auto Battler

Date: 24/09/2021

Supervisor: Jarek Francik

Project Type: BODY of CREATIVE WORK

**WARRANTY STATEMENT**

This is a student project. Therefore, neither the student nor Kingston University makes any warranty, express or implied, as to the accuracy of the data nor conclusion of the work performed in the project and will not be held responsible for any consequences arising out of any inaccuracies or omissions therein.

**Table of Content**

1. **Abstract**
2. **Acknowledgments**
3. **Introduction**
4. **Main Body**
   1. **Scope**
   2. **Development Process and Environment**

* **Tools**
* **Process**

1. Interface
2. Unit Creation and Combat System
3. Game Flow
   1. **Outcomes**
   2. **Professional And Ethical Issues**
4. **Discussion**
5. **References**
6. **Appendix**
7. **Abstract**

Auto Battler games are a relatively new game genre that came to surface into the online video game market in early 2019. Many game studios rushed to develop their own take on the game genre (Riot Games, Ubisoft, Blizzard and Tencent, etc...).

The Pokemon franchise is the highest grossing media franchise of all time (Tilemax, 2021), and has many passionate fans across the world. Many players of the game have suggested that they would like to play a Pokemon Auto Battler. So far, The Pokemon Company has not released any information on an upcoming release of the game genre.

We have therefore taken on the task of creating such a game, in hopes of pleasing the fans.

1. **Acknowledgments**

I would like to thank my parents, my brothers Amin and Zakariyya, my cousins Ilyes, Samy, Soufiane and Younes, my uncles as well as my friends Sammy and Yacine that all helped me keep motivated whilst working on the project.

Special thanks to my project supervisor Jarek and my tutor Vasileios who taught me game development.

Gratitude to OldKeith for inspiring us with his game concept.

1. **Introduction**

Auto battler consists of a strategy game where a set number of players battle each other until the last player standing is declared victorious. The game contains 2 phases, one for recruitment and one for combat. During the recruitment phase, players can purchase new units from the shop or evolve their pre-existing ones into stronger ones. They must also manage the positioning of their units on their part of the map. For the combat phase, players are combined into pairs and must face each other. Every player starts with the same health points, and loses health points when being defeated in battle. The chess component of the name comes from having a checkerboard tile map where players can position their units on. Below is a diagram displaying the game concept of Auto Chess.

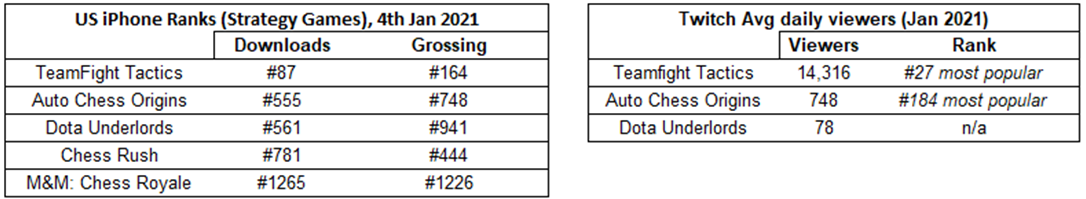


*Image explaining the game flow of Auto Battlers*

Auto Battlers(Auto Chess and Auto Battler are synonyms and will be interchanged throughout this document) attract both young and older players that are looking for a relaxing gaming experience. When asking Auto Chess players about their gaming experience, the vast majority have stated that “they enjoy the lack of micromanagement, and the demand of reflexes and dexterity” (Game Developer, 2021). As it is considered a strategy game, the recurrent players have expressed that they enjoy the fact that the game does not require their full attention and can be played whilst simultaneously doing another activity (such as watching a movie or listening to a podcast).

However, the game style contains a few drawbacks, such as not being able to commercialise itself on the eSports market. To add to the list, the game is overwhelming to new players as there are many new concepts and characters that they are unfamiliar with and expected to know in order to win the game. But the most important one has to be that the game has a low player retention (Deconstructor Of Fun, 2020), mostly due to the repetitiveness and a lack of complexity, whilst also not having much player skill expression compared to other game styles.

Many game studios have developed their own take on the game genre on PC and/or Mobile, either using a pre-existing fictional world or creating a new one specifically for their game. The most popular and economically successful ones today are the Auto Battlers that branch upon an existing lore (<https://www.appannie.com/en/>, 2021)(<https://twitchtracker.com/>, 2021).



*Tables showcasing the most popular Auto Battler games*

The Auto Chess style brings a refreshing game to players that are already interested in the franchise, and even though the Auto Battler game genre is not as popular as it once was (Deconstructor of Fun, 2020), the genre will just cement even further players’ interest in the pre-existing game lore.

Whilst playing the League of Legends version of their Auto Chess, I suddenly realised that this game style would fit perfectly with the Pokemon world. I instantly researched online to see if the game had been developed, but only found forum and blog posts from zealous fans stating that they would love to play a Pokemon version of the game. Even The Gamer, which is a gaming news website with a substantial social media following, released an article named “These Franchises NEED an Auto Chess Game” in June 2019, declaring that “Pokemon is absolutely the ideal franchise for an Auto Chess Game” (The Gamer, 2019).

Despite having a wide array of competitors in the auto chess market, we have come across only one competitor that has developed specifically a Pokemon Auto Battler (<https://pokemon-auto-chess.herokuapp.com/>, 2021). Their game is a free 2D game playable on a web browser. It includes many pokemon units from the 1st to the 3rd generation of pokemons. The game is quite similar to the one we are developing, however ours will be in 3D, and we will strive to have a more user friendly interface and game design. As you can see in the screenshot below, the user interface isn’t very intuitive.



*Screenshot of the Pokemon Autochess competitor*

We will be aiming to produce an interface in the style of the most popular Auto Chess game Teamfight Tactics, here is a gameplay screenshot :

****

*Screenshot of TFT: The current most popular Auto Chess game*

1. **Main Body**
   1. **Scope**

Auto Battlers are played on both PC And Mobile, but we have decided for our project to develop it solely on PC. We have taken this decision by considering the fact that the majority of Auto Chess players play exclusively on PC (JB-Dev.net, 2021).

Each game can last up to 40 minutes, which defy the purpose of the mobile gaming aspect of being able to play on the go. In Addition, the mobile versions seem to have a clustered user interface and hold too much information during play time considering the size of a mobile screen. Overall, the game genre “has suffered quite a bit in the transition to the smaller screen” (Venture Beat, 2019). As shown in the screenshot beneath, there are lots of UI components to keep track of during the game for such a small mobile screen.



*Screenshot of Autochess Mobile and its clustered mobile UI*

The game design and balancing was inspired by OldKeith, a Reddit user who created a pokemon autochess game concept and generously let us use his game idea for our game (Reddit, 2019).

To develop and design the game fully would take longer than the time we were allocated, so we have discussed with our supervisor and agreed that a playable prototype would suffice. Therefore, having a game with working mechanics was prioritised over game design and game balancing. Below are the requirements specification that we came up with in the early stages of development:

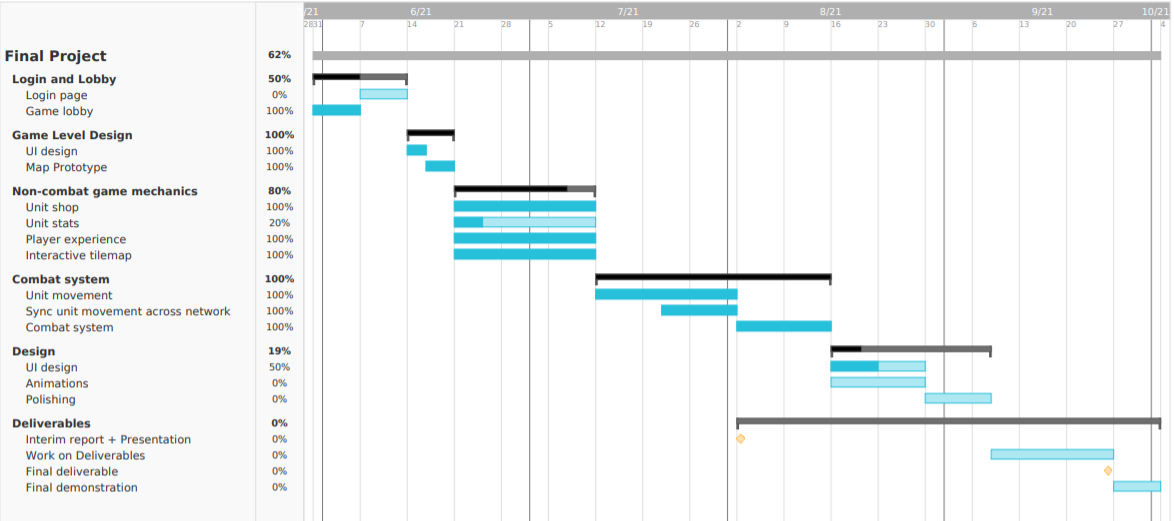
* + Each game could be composed of two to eight unique players.
  + The first three rounds should be player vs environment rounds where the players face NPCs (Non-player character). The subsequent rounds will be player vs player, with a possibility of an occasional PvE round here and there.
  + The unit bench could have ten slots where the players’ units that are not on the board will be placed.
  + Players must be able to purchase new units from a shop with randomly selected units.
  + Purchased units must be automatically placed on the bench, or on the board if the bench Is full. If both are full, the player must not be able to purchase a new unit (unless that unit results in an evolution).
  + Players must be able to sell their units for a price lower than the purchased one.
  + Players must gain some experience per round and also be able to purchase experience from the shop.
  + The number of units being able to be placed on the board will be based on their level (up to a maximum of nine).
  + Players must be able to freely place their units on their part of the board (as long as the number of units on the board doesn’t exceed the player’s level).
  + Upon purchasing three of the same unit, the unit should evolve into its designated evolution (up to a maximum of two evolutions).
  + Players should be able to view the opponents’ units that have been placed on their board.
  + The combat system must be fully automated, units placed on the board should attack the closest enemy unit unless they are ranged units.
  + Units must be able to move forwards, backwards, left, right and also diagonally on the board.
  + Units will start each round with zero mana, and gain mana each time they auto-attack. Upon reaching maximum mana, they will use their special ability.
  + Each player’s health points should be shown on the game’s UI.
  + A player will lose a small amount of health points if they lose the round.
  + Each unit’s health points could be shown using a health bar.
  + Every unit must have their health points restored to the maximum at the start of every round.
  + Having a given number of units that are from a type or class should give benefits to all pokemons of that type/class (For example, having three pokemons of Water type should increase all water damage by 30% to the water type pokemons).
  + The winner will be declared when all other players have lost all their health points.

The project was proposed on the 28th May 2021, and development started the following monday (31st may 2021). The hand-in deadline was on the 24th september 2021, giving us seventeen weeks in total to complete the project.

We started the work with creating a game lobby and designing a simple user interface alongside the level design. Following that was the development of non-combat game mechanics, such as buying and spawning new units, placing them on different tiles, as well as unit evolution.

The biggest portion of time was allocated to creating a movement and combat system for Pokemon units. As stated in the requirements above, we had initially planned to have the units move on a hexagonal tile map, but after spending too much time on it without success, we had to continue with the rest of the development, and we therefore let units move freely on the terrain.

The final three weeks were used to polish the game and to create the required deliverables. The following Gantt chart visually represents the project plan discussed prior.



*Gantt chart showcasing the project timeline*

* 1. **Development Process and Environment**
* Tools

The product was developed on Unity 2020.1.17. It is the game engine we were the most familiar with, and Unity contains a user-friendly workstation that allows the game level to be created visually. The game balancing and tweaking on Unity can also be produced interactively without having to access the code. The game mechanic scripts and other game functionalities were programmed using the C# language on the Visual Studio 2019 IDE (Integrated development environment).

For the networking, we have used a free version of PUN v2 (Photon unity networking version two). It is a multiplayer networking package allowing us to create new game rooms, but also syncing the game state across the clients.

* Process

The graph below shows a simplified version of the project architecture that includes most mechanics found inside the game. It contains a GameController to manage units as well as a GamePlayController that handles the flow of the game. The Board controllers store the bench, player-owned and enemy tiles of each board, whilst the Pokemon Controller handles the combat and movement of units.

Table

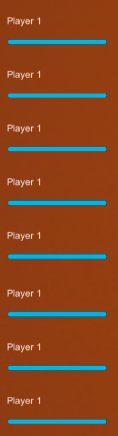
Description automatically generated with medium confidence

*Simplified Class Diagram of the project*

1. Interface

For our user interface, we were aiming to produce one similar to other competitors,

as they seem to be the most intuitive. The player’s name and health bars are displayed on the right-hand side of the screen. Whenever a player loses a battle, his health is reduced and this is updated to all other clients with the use of an RPC.



*Screenshot of the player list and their health bars.*

Each player will have its health bar color changed to yellow alongside a “(YOU)” text to indicate which player they are in the player list.



*Screenshot showing the player’s health bar highlighted in a different colour*

On the left is where the pokemon types and classes of the units placed on the player’s board alongside their counter are presented.

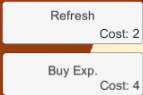
The unit shop is located at the bottom. It includes five image buttons that each represent a random Pokemon that the player can purchase if he has enough gold.

On each button is displayed the type and class of the pokemon, alongside its name and cost. Once hovered, the image button is highlighted in blue.



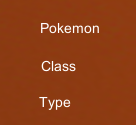
*Screenshot of the Pokemon unit shop*

There are also two buttons, one for refreshing the shop with new random pokemons, and one to increase the amount of experience of the player. But note, these come at a cost, which are showcased within the respective buttons.



*Screenshot of the 2 buttons*

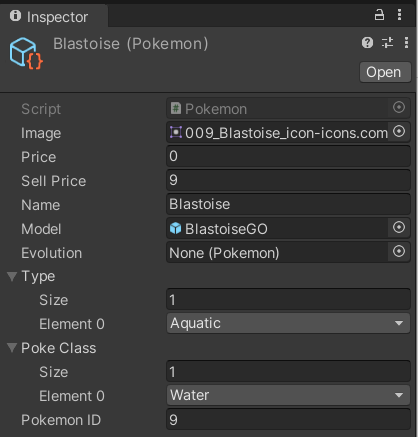
Hovering over one of your units will display its name, class and type on the bottom right of the interface.

**

*Interface displayed when hovering over a unit*

1. Unit Creation and Combat System

Our database of pokemon units was populated thanks to the fabrication of a ScriptableObject script, allowing us to create new Pokemons on the Unity Inspector thanks to the addition of a “[CreateAssetMenu]” tag (See appendix A for full script). According to Unity’s documentation, “a ScriptableObject is a data container that you can use to save large amounts of data, independent of class instances. One of the main use cases for ScriptableObjects is to reduce your Project’s memory usage by avoiding copies of values. This is useful if your Project has a [Prefab](https://docs.unity3d.com/Manual/Prefabs.html) that stores unchanging data in attached MonoBehaviour scripts … you can use a ScriptableObject to store the data and then access it by reference from all of the Prefabs." (Unity, 2018)



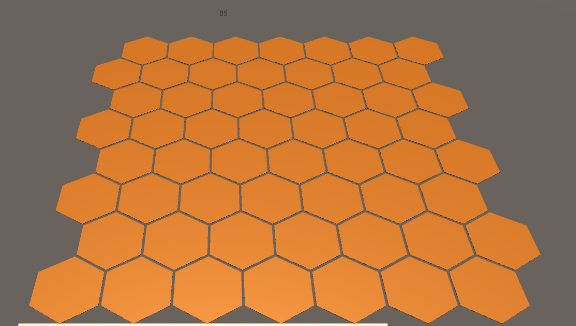
*Screenshot of the creation of a Pokemon object within the Inspector*

The pokemon 3D models were sourced from the Pokemon XY series games resources website (<https://www.models-resource.com/3ds/pokemonxy/>, 2021). They were rescaled to match the correct game dimensions, and given multiple components, such as a PokemonController in charge of controlling the movement and attacking of the unit, a NavMeshAgent permitting the unit to move on the map and a PhotonTransformView to sync its movement across the network.

The unit shop is composed of five image buttons that each represent a purchasable Pokemon. Upon clicking on either of the five shop buttons, the Pokemon that was assigned to that button will be instantiated and spawned at the player’s first available spawn point (the spawn points are composed of the player’s bench tiles and also the tiles that are on the player’s side of the game map). To keep the game state synced, and since unit purchases are not something that are getting called every frame, we have used an RPC (Remote procedure call) that gets called from the buyer to all other players in the room. That RPC takes 2 parameters: one for the playerID of the buyer and another integer for the ID of the pokemon. The function then checks to see if the pokemon in question is ready for evolution. If so, it will remove the 2 existing pokemons and spawn the evolution. Upon instantiating the unit, the tilePosition property of the PokemonController component is assigned to the tile that it was spawned on.

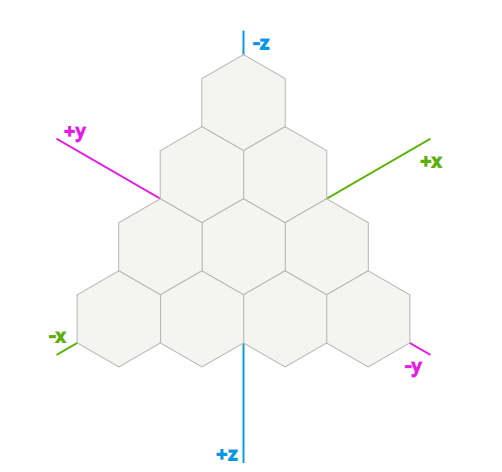
When a player holds down the click on one of his units, he can drag the unit across the screen, and reposition it on a different tile if he likes.The repositioning of units was done thanks to the pre-existing Unity functions OnMouseDown(), OnMouseDrag() and onMouseUP(). Upon clicking on that unit, the selectedPokemon property of the GameController script is assigned to the current object in question, and the unit is elevated slightly upwards to indicate that the unit is selected. Moving the mouse will make the unit follow the cursor, this was calculated using the ScreenToWorldPoint() function on the mouse cursor. Upon releasing the click, the script checks if a tile was selected. If the tile in question is one of the tiles on the map, it will investigate if the player has permission to move the unit on that tile (if his level is high enough). If the placement is allowed, an RPC is called to all players to reposition the unit on the appropriate tile, this is done so players can keep track of what their enemies have on their board and to keep the game synced.

The creation of the tile map was done by looping through two nested for loops, the first one looping through the predefined map height and the second one looping through the map width. A new gameObject was instantiated and positioned properly thanks to the tile offset passed through in the parameters, and each tile was given a Node object with the current for loop indices as the coordinates.



*Screenshot of the Hexagonal tile map*

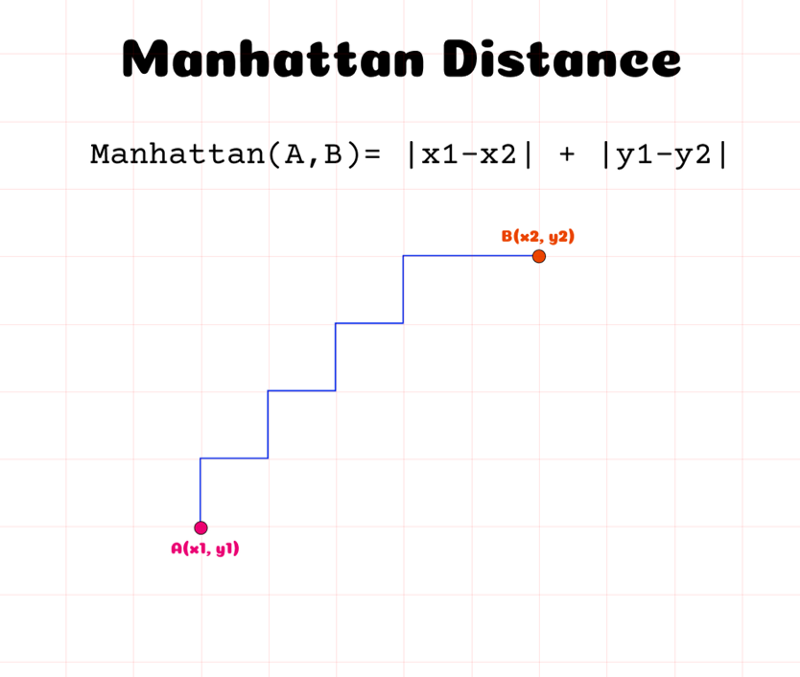
Initially, we wanted our movement system to allow units to only move on unoccupied hexagon tiles. We spent a few weeks developing this feature. The movement algorithm used was an A\* pathfinding algorithm. Since our map was composed of hexagonal tiles, we had to find a way to convert the 2 Dimensional coordinates into cube coordinates (3 Dimensional).



*Image showcasing cube coordinates on a hexagon map*

This was done by the following function (Red Blob Games, 2013), which is found in Appendix B.

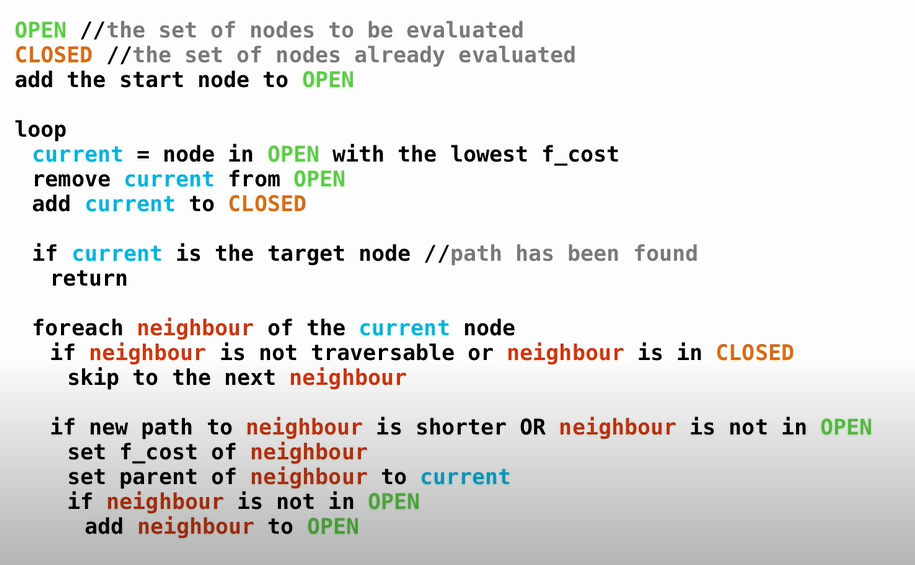
We used a Manhattan distance function (Red blob games, 2013) to calculate the distance between 2 nodes, which was customised to use cube coordinates.



*Image explaining briefly the Manhattan distance*

The function responsible for calculating the distance is found within Appendix C.

The pathfinding algorithm is described below in pseudo-code (Sebastian Lague, 2014, 7:42)



We managed to get our units moving and following their target with some customization of the algorithm. However, when adding the script to both the target and enemy, we stumbled upon some complications. Since the pathfinding algorithm was being called in the Update() function (at every frame), it had trouble keeping track with which nodes were occupied and which weren’t. Therefore the bug we encountered was some units were blocked in between tiles. Due to the time constraints of the project, we had to give up on implementing the pathfinding algorithm within our game.

We then chose to allow units to move freely on the map, with the help of a Nav Mesh Surface being baked on the terrain and a Nav Mesh Agent for each unit. Firstly, the unit will look for the closest enemy, and assign it as its target. Afterwards, the unit will move towards that enemy until in attacking range, where it will start attacking the target.

The attacking system is composed of two types of attacks : one basic attack, and one special attack. Basic attacks deal small amounts of damage, and give the unit some mana. Once the mana bar is full, the unit can then perform his special attack, which deals a greater amount of damage. Every unit owns two slider bars to showcase the state of the health points and mana points.



*Screenshot of a Pokemon unit with its health and mana bar.*

Both the movement and attacking are controlled by the master client, then the movement is synced to other clients thanks to a PhotonTransformView. The attacks are timegated by the attack speed of the Pokemon unit. Once the timer reaches zero, the unit will either perform a basic attack or a special attack if his mana bar is full. This is also performed by the master client and sent to all other clients thanks to the use of an RPC that passes the unit ID of the target as a parameter.

1. Game Flow

The game in itself consists of two stages, which we defined in an enumeration including 2 variables : a Preparation game stage and a Combat game stage.

The preparation round lasts for forty-five seconds whilst the combat one lasts thirty seconds. A timer counts down, and upon reaching zero the master client will send a call to other clients to change the game stage and do the appropriate configuration of units and the board.

If the gameplay controller is in the combat stage, the script performs a check on the round number. If the round number is less than three, players face NPC-owned units, which is essentially a PvE (Player versus environment) round. If greater than three, the master client will pair enemies to fight each other. If there is an odd number of players, an extra NPC is added to the list, whom one random player will face. That NPC is given a few units to face his enemy. The function responsible for this process can be found in Appendix D.

There is then PlayerList.Length/2 pairs that face each other, where the second player in the pair will have its units placed on the board of the first player’s board. Units are placed in the exact position that the player placed them in, just on the opposite side. This is done thanks to the function in appendix E, which takes the current tile Node of the unit, and looks through the enemy’s Nodes by subtracting the X and Y coordinates to the width and height of the map minus one.

Once the combat winner is decided, the loser will see a reduction in his health points, and this is forwarded to all other clients thanks to an RPC. All players are given extra gold and experience at the end of each combat round.

If the gameplay controller changes to a Preparation stage, all units are placed back on their original position, their health and mana points are reset and the camera of the player gets positioned back to his board. Once the forty-five seconds of preparation are up, it is time for another combat round.

Once a player sees his health points reach zero, they are eliminated from the game, and the game will check to see if there is only one player left standing. If so, he is declared the winner, and will be prompted to return to the game lobby.

* 1. **Outcomes**

As mentioned prior, we were told by our supervisor to complete as much work on it as possible, and to at least come up with a playable prototype. Due to the nature of our course, we therefore prioritized the networking processes, the game mechanics and a working interface over game design and game balancing.

We managed to create a playable prototype that encompasses most game mechanics found in popular Auto battlers. That includes unit purchase (and spawning), selling, evolution, repositioning amongst many things. For the combat stages, the units placed on the board would fight each other until a winner is declared. The game controller would handle player’s health points and game stage transitions.

However, two game mechanics that were lacking that we had hoped to implement at the beginning of the project were class and type bonuses of units boosting their damage or stats, as well as itemization that were gained during PvE rounds and placed on the player’s units.

The level design we used was the one we briefly made in the early stages, which is just composed of a plane, on which fifty-six hexagonal board tiles and nine squared bench tiles are present. We were glad to find authentic Pokemon 3D models that were usable in our game, however we did not have the time nor skillset to give these models walking and attacking animations. For our User interface, we used the preset UI objects from Unity, which we edited slightly to match our needs. This gave the whole interface a rather bland and unprofessional style.

Overall, you can tell the game design is still lacking some work, and has a prototypical look. We hope that with future work the game will be able to be enjoyed by fans.

* 1. **Professional and Ethical issues**

The project does not pose any security issues since it is a video game that is played on a computer, therefore not putting the players at any risks. Since we did not allow players to create their own accounts using their credentials, there was no sensitive information being handled in the game. The video game we developed does not cause any ethical issues, as it is simply a strategy-based game. Many games of that style have already been published, none of them causing any ethical issues.

The game made use of Pokémon assets. Because this will cause many legal issues with the Pokémon producers and publishers, we do not plan to release the game publicly on video game stores or video game sharing websites. It will simply be shared privately to friends, families and online fan communities.

1. **Discussion**

So far, we have not shared the game with friends or avid fans, as we believe the game still requires some work in order to be fully playable.

The strengths of the project include game flow and unit management (purchasing, evolving, repositioning) as well as the hexagon tile map movement. Even though we were not able to implement it in the game, with future work we hope to incorporate it and for it to become a highlight of the game.

The game still contains a few bugs and as discussed prior the design still requires some work, including game balancing, which were due to lack of time. The plan was followed appropriately, however we spent more time than expected on developing the hexagon tile map movement, which delayed everything by a few weeks which could have been spent on game design.

For the future, we hope to implement unit combat and movement animations, extra game mechanics such as itemization and also unit class and type bonuses. Also the combat system could be developed further to include a more refined damage calculation function that takes into consideration the stats of units, as seen in the original Pokemon games. The User interface could be changed to become more interactive, such as animations when health bars’ values are changed or when buttons are clicked on. All the UI elements could be styled in a way that matches with the Pokemon franchise.

1. **References**

Tilemax (2021) *The 25 Highest-grossing media franchises of all time.* Available at: <https://www.titlemax.com/discovery-center/money-finance/the-25-highest-grossing-media-franchises-of-all-time/> (Accessed: 21st September 2021)

Game Developer (2021) *AutoChess: Market status and Design Analysis.* Available at:<https://www.gamedeveloper.com/disciplines/autochess-market-status-and-design-analysis> (Accessed: 21st September 2021)

The Gamer (2019) *These Franchises NEED An Auto Chess Game*. Available at: <https://www.thegamer.com/auto-chess-game-franchises/> (Accessed: 21st September 2021)

Deconstructor of Fun (2020) *Why AutoChess games can’t monetize - And how to fix that.* Available at: <https://www.deconstructoroffun.com/blog/2020/1/28/2020-predictions-7-auto-chess?fbclid=IwAR1lII_37VYWi2YA6FYfKFvxTa_JpEEjnItB2efr8yxIj1zuLJLDoNi0LhM> (Accessed: 21st September 2021)

<https://www.appannie.com/en/> (2021) (Accessed: 21st September 2021)

<https://twitchtracker.com/> (2021) (Accessed: 21st September 2021)

<https://pokemon-auto-chess.herokuapp.com/> (2021) (Accessed: 21st September 2021)

JB-dev.net (2021) *Autochess Reddit players poll*. Available at: <https://docs.google.com/spreadsheets/d/1f-Xtp7MkS6bHkYZfS5KWbb2FILN-DR5133BqTvpifHg/edit#gid=845416165> (Accessed: 21st September 2021)

Venture Beat (2019) *Auto Chess on mobile is improving… But confusing*. Available at:<https://venturebeat.com/2019/05/30/auto-chess-on-mobile-is-improving-but-confusing/>) (Accessed: 21st September 2021)

Reddit User OldKeith (2019) *Pokemon Autochess [Concept],* Available at: <https://www.reddit.com/r/AutoChess/comments/ar4cjh/pokemon_autochess_concept/>(Accessed: 21st September 2021)

Unity (2018) *ScriptableObject*. Available at: <https://docs.unity3d.com/Manual/class-ScriptableObject.html> (Accessed: 21st September 2021)

<https://www.models-resource.com/3ds/pokemonxy/> (2021) (Accessed: 21st September 2021)

Red Blob Games (2013) *Hexagonal Grids.* Available at: [https://www.redblobgames.com/grids/hexagons/](https://www.redblobgames.com/grids/hexagons/#coordinates)(Accessed: 21st September 2021)

Sebastian Lague (2014) *A\* Pathfinding (E01: algorithm explanation)*. December 2014. Available at: <https://www.youtube.com/watch?v=-L-WgKMFuhE&ab_channel=SebastianLague> (Accessed: 21st September 2021)

1. **Appendix**

A: 

B:



C:



D:



E:

****