The Conceptual Model & Tools Used

To provide an abstract representation of the architecture of our team's software, we will be using the UML 2.X specification. We will be using this specification as it will allow us to create a clear and basic diagram of the game and how it should be created. The tool used to create the architecture representation is Lucidchart, we have chosen this tool because of its ease of use as well as the collaborative convenience it offers.

Before modelling the conceptual design of our software's architecture, we first had a group discussion as to how the game would work in the real world. We started by contextualising the game through simple scenarios and jotting down any ideas on the whiteboard, focusing on how the game would work and respond to each one. This ensured that all team members understood the requirements of the user as well as helping us decide the main classes that will be used in our system.

We quickly realised that using any tools might force a low level of abstraction as they require detailing like attribute types etc. which may make it harder for us to sketch an abstract model for our system. So, we initially began to design a very simplified UML diagram on pen and paper as shown in Figure 1. Using a class diagram ensured all team members understand the basic structure of our software. The rough sketch showed only the classes and the high-level relationships between them which allows us to visualise the game's structure in very simple terms so we could use it as a base for our final UML diagram.

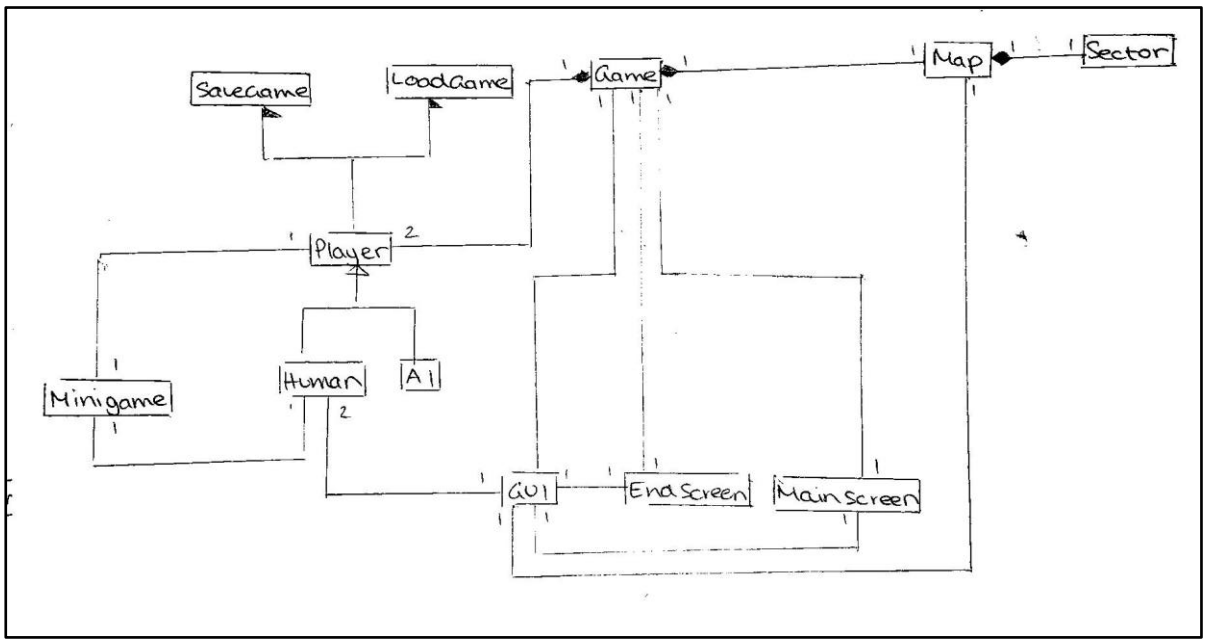
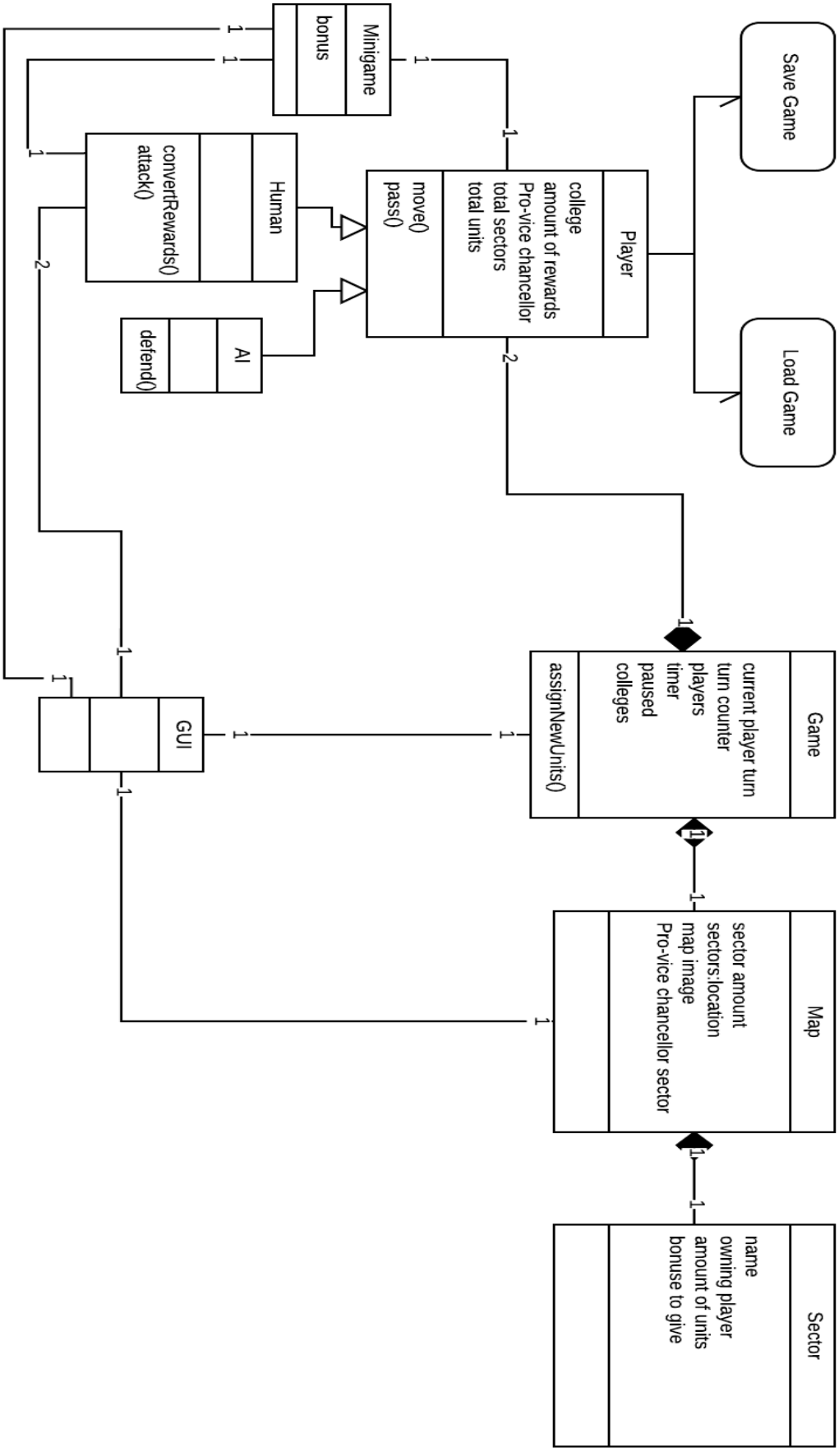


Figure 1. A scanned copy of our abstract UML sketch

The sketch allowed us to move on to creating a much fuller UML diagram that we can use for referencing in the later stages when implanting the game. It extended upon the main attributes of the generalized entities we created in our previous diagram.

Figure 2. UML class diagram for to the conceptual architecture of our system



Justification

We have decided to use an abstract UML class diagram as it gave us the capability to go into enough detail to cover all of our requirements without going into too much detail keeping abstract.

Game

The game class is a representation of the current game in progress. This will store all of the necessary information of the current game state. This includes the state of all the sectors; The information contained to each player in the game along with the neutral AI player; The current turn of the game; What the game time is at to display to the user as well as whether or not the game is paused by either of the players.

At the start of the game, each sector will be randomly allocated to a player or the AI (neutral college) (F6.1). The game class should be responsible for this as it has access to both the player class and the map class. This class can then assign the sectors to the owning player. During this a larger quantity of sectors should be given to the AI (F6.2). The PVC sector should be randomly selected from the sectors that are claimed by the neutral college (F3.2). When the player has found the PVC sector, it should be announced to all players that the PVC sector has been found (F3.3). This should be controlled by the game class as it has access to the map class to see all of the sectors to assign the PVC sector as well as to announce to the other player when a player has found that sector.

During play a player will inevitably want to attack one of their competitors, this will require a system to work out who wins the fight. If it's the attacker, they win the sector (F9.2) and need the loss to the attacking units to be calculated (F9.1). Whereas if the defender wins, only the loss to the units need to be calculated. This should be controlled within the game class as it will be able to access both of the players classes and the map class so can read what units are attacking and calculate the outcome from that. A player may wish to pause the game during the turn to prevent the timer from increasing (F2.3). This will keep their end of game statistics as accurate as possible. As the timer is in the game class the method to pause should also be within the class.

The winning conditions of the game should be checked on a very regular time, almost every action (F12.1). As an attack could wipe out the last college winning the game or the timer could expire, this means that game should output the statistics of the game, and again as the game class has access to all the data it is the most apt class to do so.

Map

The map class contains all of the sectors along with all of locations of these sectors. This includes the PVC sector (F3.1) which should be stored with the rest of the sectors as well as on its own in the map class for easy reference. It should be checked against later to see if the player has found the PVC sector. The Map class was the best choice as it stores all the other sectors.

Sector

The sector class embodies just a single section of the map. The sector will contain a name, if it is a landmark of the University. It will also keep track of which player owns the tile along with the amount of units that have been placed in the sector. Each sector has a bonus to give whether it is something or nothing, the sector will keep track of that too. The bonus for capturing the PVC sector will be a greater bonus (F3.4) than any other sector in the game this will be stored within the sector class, as with other sectors, the bonus is gained on the capture of the sector. The sector class will store who owns it (F9.4) as during any unit movement it's simpler to then change the owner once the sector is captured. Each sector should know how many units are on that sector as each fight or unit movement can then edit that number (F10.1). This also applies for when any units are added at the beginning of a player's turn when they receive new units (F10.2). Each sector should be aware of it's name if it's a landmark as this is displayed to the player (F11.1).

Player

The player class contains all of the information relevant to each player. This will be different for each player as they will be at different stages within the game. For example one of the players may have gathered more sectors than the other, therefore will have a higher amount of rewards because of that. They will also almost always have a completely different amount of units and sectors. The player class has two subclasses being Human and AI. This is because the AI will be automated and won't be able to attack as it can only defend its own sectors. This means that the AI will have methods which allow it to defend its own territory. Whereas the human player will try to take sectors from their competitors and this will be done through actions the human will perform.

At the start of the game the player should be able to decide which college they want to play and the player will store which college they are in their class as it's respective to them (F5.2).

Each player needs to have the option to save and load a game (F2.1, F2.2) so the player is the best class to handle this as it is an action the player makes.

During the game, players will perform actions such as attack or move units. These actions are shown to the user and then the user must pick an action to perform (F8.1). During an attack a player may lose some of their units and this should be updated within the player class as well as the sector as the player should know their total amount of units (F9.3).

At the start of each turn the player will receive more units (F10.2) and the players should be able to decide which of their sectors the units go in, again this is an action that the players perform. As well as this the amount of units that they get at the start of turn should be calculated within the player class (F7.1) as the total amount of sectors that the player has are stored there.

AI

The AI class is a special type of the player class as it won't be able to attack, meaning it can only defend (F6.4), or gain any bonuses of any kind. The AI will also not gain any extra units at the start of its turn(F6.3). The AI serves as an obstacle for the players and is a key part of the game.

Minigame

The minigame class will contain all of the necessary information to start the minigame which is unlocked via the pro-vice chancellor sector. The minigame will have a bonus to give to the player and therefore will have to have the information pertaining to its bonus (F4.1).

GUI

The GUI class will contain all of the methods and attributes needed to display the game onto the monitor so that the players can interact with the game. This includes the start screen, the end screen, the map and the minigame as well as any menus (F11.2).

They key features that have to be displayed during the playing of the game are the choice of colleges at the beginning of the game (F5.1). The map should be displayed so that all of the sectors are identifiable to each college for the players to easily see which one is theirs, be it by a different colour or logo (F5.3). It has to show the units that the player has to the player as well as how many units they gain by converting their bonuses to units (F7.2). The options that the player can perform have to be shown to the players at all times when they can indeed perform the action (F8.1).

The map has to be clearly separated into sectors that can the borders can be identified easily (F11.1). Each state has to be able to show any valid information about itself to the user.