**CS-230 Software Engineering**

A1 Object-Oriented Design Document

**2.1.1 Candidate Classes and Responsibilities**

|  |  |
| --- | --- |
| **Player** | |
| Responsibilities | Collaborators |
| inventory (Holds the tokens, keys and other items) | Cell  Collectables  Level  Enemies  LevelSelect |
| Restart Check (If Player lands on a kill cell then new instance of player (resetting inventory) (in Profile) |
| goal (ends the game to then be presented with |
| move (manipulation of direction and cell moves within level it will check what cell the next move will land on then run that event (teleport etc.) |

Player-This will hold the attributes that the player holds to use within each level, it will respond to different events like teleportation, goal, kill calls, doors etc.

(Subclass of Moveable) (Author- Ciara and Bryony)

|  |  |
| --- | --- |
| **Profile** | |
| Responsibilities | Collaborators |
| Current Level (Based on levels completed (linked list implementation)) | Level  LevelSelect  MainMenu |
| Store Name of profile |
| Hold save (Holds the save file of the player that has left midway through the game) |

Profile- These are the instances of profiles that users will have to record their current progress in the level, along with their high scores. There will be multiple instances of Profile so that users can compete against each other for the highest score. This could also be used if we were going to save progress of a level if the user quit before finishing, as we could save their current location and attributes to the Profile for when they log back on.

(Own Class) (Author- Ciara and Johannes)

|  |  |
| --- | --- |
| **Cell** | |
| Responsibilities | Collaborators |
| Abstract Class all cell types inherit from cell cords/location (x , y) linked list | MapReader  Moveable |
| Adjacent Cells (north, east, south, west) |
| Cell Type (The type of cell) |
| Is cell solid (can it be walked though for enemies, players can go to individual classes that inherit from cell to see if they meet criteria to pass) |

Cell- This an abstract class that will hold all the different cell types, it mainly holds the type and the position as well as adjacent cells.

(Super class of Ground and Wall) (Author- Ruth and Jake)

|  |  |
| --- | --- |
| **Smart Enemies** | |
| Responsibilities | Collaborators |
| Position (holds the position of itself) | Level  Player  Cell |
| Player Position (holds the players current position (gets from level class)) |
| Shortest Route (Calculates shortest route from its current position to players position) |
| move (moves into the next cell only if it isn’t wall cell for inherited enemy class) |
| If no path available to player, then moves to random valid cell |

Smart Enemies-This class will only hold the characteristics of the smart enemy, as it’s a subclass of Enemies. Its main function is to move along the shortest route possible to the player so that it can kill it.

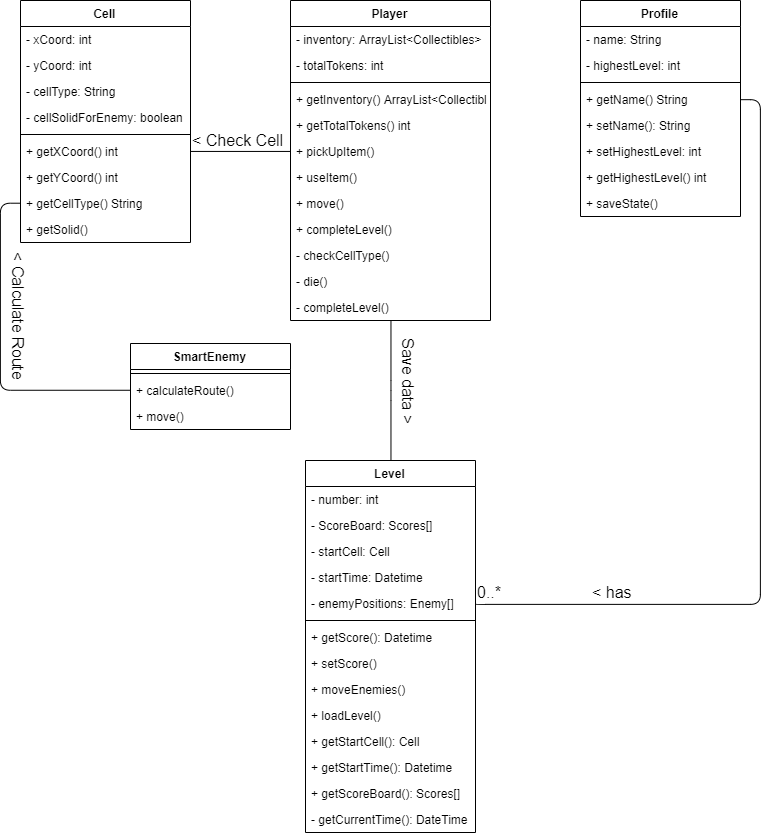
(Sub Class of Enemies (Subclass of Moveable)) (Author- Tom and Rohini)

|  |  |
| --- | --- |
| **Level** | |
| Responsibilities | Collaborators |
| Level Name (e.g. level1) | LevelSelect  Profile  Player  Enemies  MapReader |
| Top Score (Stores the top 3 scores for each instance of level and checks that if the score has been beaten) |
| Keeps track of what is within the level and location (gets the enemies etc.) |
| Load level (this starts the time and the screen to present the level with the current positions) |
| Holds the current score and also can change the start score counter to whatever was stores if file was saved midway through level. |
| Saves player to profile every move player takes |

Level-This is the class that will hold all the data for the level, making it easy to save the current progress. It will also hold the 3 highest scores, and the locations of all the Moveables.

(Own Class) (Author- Johannes)

**2.1.2 Class Diagrams**



**Collaborations**

**Profile-Level –** Each instance of a profile stores the information about the completed levels. If no levels have been started, this data will be empty, and no instances of Level will be in Profile. As levels are saved/completed, the Profile will update to show if the Level is complete or not, the score, and will store this.

**Player-Level** – When the Level is saved, all the information and data about the level must be saved, such as the score and the positions of each enemy. As well as this, the player position must be saved as well as the inventory of the player.

**Player-Cell** – When moving, the player must perform a check to see if the cell that they attempt to move into is valid. The cell will then return its type and the Player class will handle the data accordingly. For example, if the player attempts to walk into a Wall cell, they will not move, as they can’t move into it.

**SmartEnemy-Cell** – To calculate the shortest route to the player, the enemy must check each cell for their type and the adjacent cells in the direction of the player to find a path. Each cell would return the relevant information until the shortest route has been found.

**Complex Behaviours**

**SmartEnemies.CalculateRoute()** -This method should find the shortest path between the smart enemy and the player, this should be constantly changing as player moves around the grid. This will be implemented by using the shortest path algorithm. This class will need to collaborate with player to find the player position and map reader to be able to find valid routes for the enemy to go e.g. not through walls etc.

**Level.Load()**-This method loads in the level from the file where the game info is stored, this could be a previously saved level which is now being resumed by the player. The load method will read in the ASCII text file so it can be printed to the screen. Load game will need to collaborate with profile to produce the grid with the objects in the correct positions.

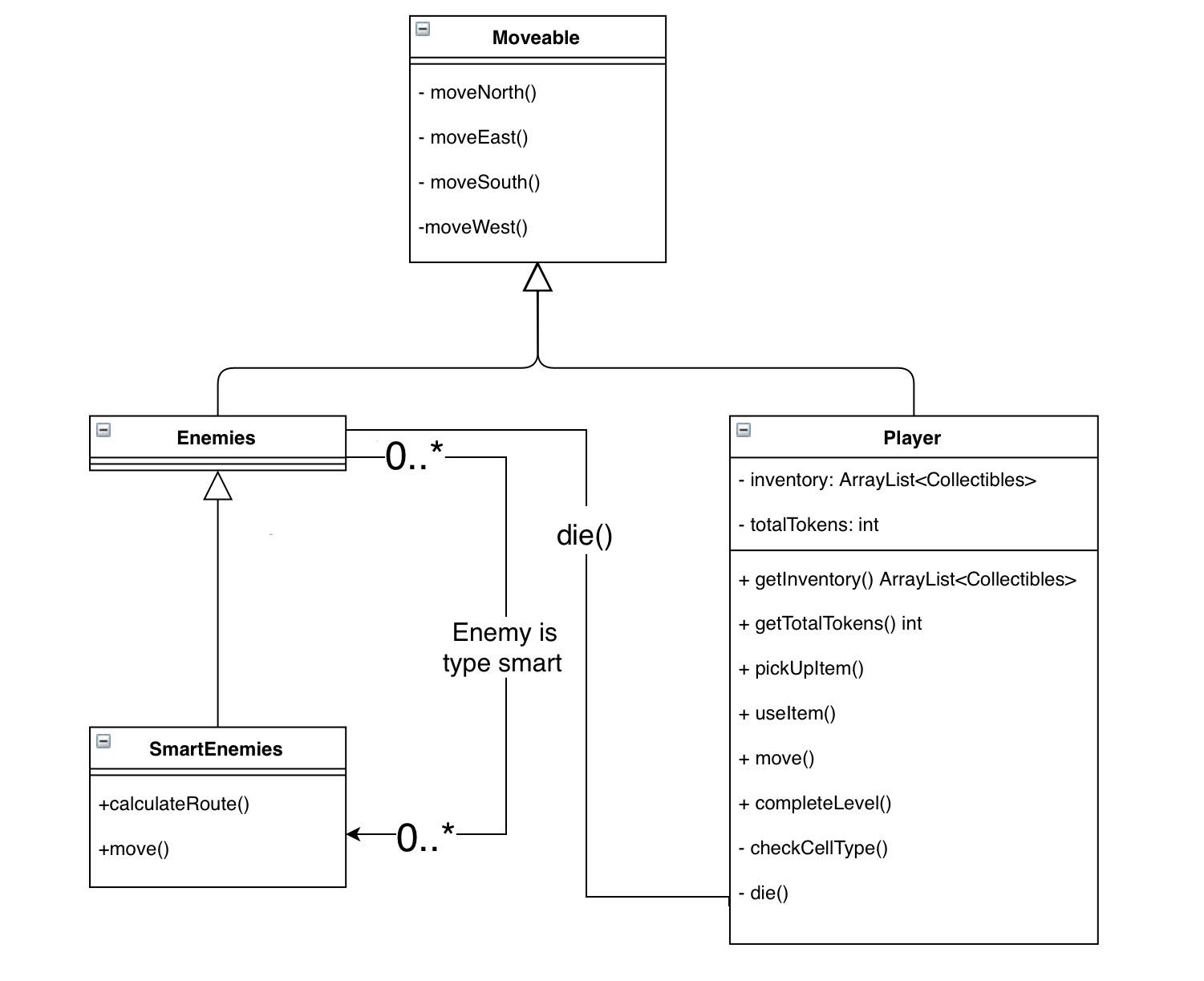
**Level.CompleteLevel()** – This method should be called once the player has successfully reached the goal. getScore should be used to calculate and save the players score for the level. The player’s inventory should be cleared and their total number of tokens reset to 0. HighestLevel should be incremented using setHighestLevel. The scoreboard should be displayed, then level menu.

**Player.Move()** – Dependent on keys pressed by player, call move up/down/left/right appropriately. Call checkCellType to determine what type of cell the player wants to move onto. Dependent on cell type call appropriate methods to use items from inventory such as keys to open doors, move the player onto the cell therefore changing their coordinates, or kill the player for example.

**Profile.SaveState()** - This is a method that will be used if the Profile exits the game midway through the level, once they exit the current level then the instance of the player will be saved to player as well as the level as you need to keep the enemies position and the tokens picked up. Saving these to profile will mean when the person goes back onto their profile, they will then be able to load their game at the currently saved place, making sure all the entities are in the same place as before so no bugs to exploit.

<EXTEND THESE IF YOU WISH, MOVE AND COMPLETE LEVEL SHOULD BE EXTENDED>

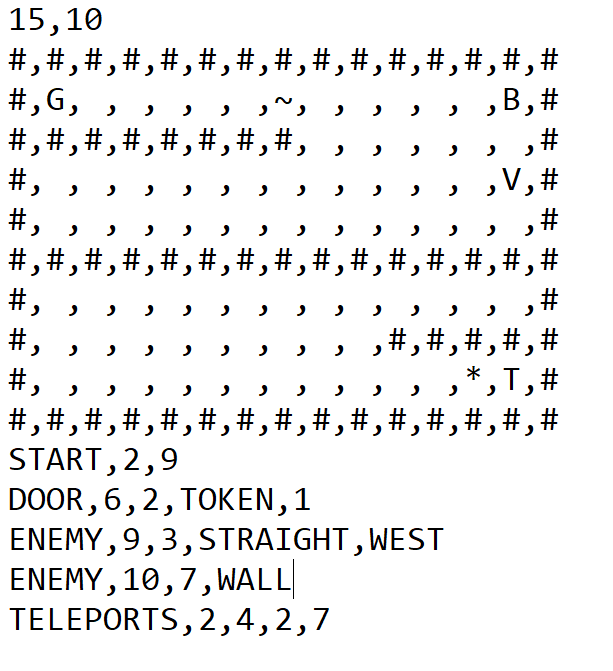
**2.1.3 Hierarchy Descriptions**



<BENEFITS OF HIERACHY AND WHY THAT DESIGN CHOICE>

We have chosen to make player and enemies subclasses of moveable. This logically makes sense because they are the only objects in the game that move. They also always move at the same time. The directed move methods in moveable should just change the coordinates of the moveable so it moves to another adjacent cell in the game. Enemies is an abstract class because all enemies are of a specific type that dictates how they move around the game. Therefore, enemies should never be instantiated. All subclasses of enemies, and player should override a moveable method move(). This method dictates how the moveable moves via conditional statements, algorithms etc. The benefit to having this particular hierarchy with moveable as a parent class, is that we expect during implementation more methods will be found and written which will be used by enemies and player, meaning they can be inherited from a parent class making code more efficient. Enemies only collaborate with player when they move onto the same cell in the game, killing the player.

**2.1.4 Level File Format**

 The first line of the level file defines the height and width of the map using the two comma separated values. Line 2 till line height of map+1 will be an ASCII representation of the cells on the map that do not require extra data to define.

Wall (“#”)

Water (“~”)

Goal (“G”)

Tokens (“T”)

Fire (“\*”)

Flippers (“V”)

Boots (“B”)

Ground (“ “)

Ground cells are replaced by any cells that are defined later.

The next line after the map shows the start position “START” of the player, represented by two comma separated values (x and y coordinates).

After this, any extra cells that have not been defined are done so, this is done by specifying the type of cell followed by any relevant data in the comma separated list.

For door this would be “DOOR” followed by the x and y coordinates, and then the type of the door. If the type is token, the token cost comes next, otherwise the colour of the door comes next. Teleporters “TELEPORTS” are defined in pairs, with the x and y coordinates of each as the data in the list. The key is a “KEY” followed by its x and y coordinates and the colour of the key. Enemies come last, with a symbol of” ENEMY” denoting them. They have x and y coordinates followed by the type of enemy. Straight Line enemies also need the direction and something to tell what side it is following

<EXTENSION NEEDED HERE AS DOCUMENT SAYS MAXIMUM 2 PAGES AND WE HAVENT EVEN HIT ONE>