# **gdrvCS-230 - A1 Partial Design Document - Group 20**

## **Lost In Space** (Alien Spaceship Theme)

**1 - Candidate Classes and Responsibilities**

|  |  |
| --- | --- |
| **User Profile** | |
| **Responsibilities** | **Collaborations** |
| Store Username and Highest level achieved.  Select profile.  Create profile. | Leaderboard |

**Class Description:** User profile will store users information such as their usernames and the highest level they have achieved. By storing the information, players would be able to select their existing profile.   
**Authors:** Kelsey Pyne (976805), Joseff Pugh (975656)

|  |  |
| --- | --- |
| **Level** | |
| **Responsibilities** | **Collaborations** |
| Knows the player’s Character.  Knows the map of *Cell* objects.  Knows *Enemy* objects present on level.  Knows the time at which the level began.  Knows total time spent before the last load.  Runs game turns from its current point until the player’s character dies or wins or when the player decides to quit or save and quit.  Sends necessary information to the Leaderboard class to update it if a player wins.  Runs a full turn, based on the character’s movement.  Remakes itself when the character dies or wins.  Creates a list of all the information needed to put itself into a save file.  Draws the UI for the player including everything to do with what's around the character and what collectables they have. | Character  *Cell*  *Enemy*  Leaderboard |

**Class Description:** Level objects hold all the information necessary to run a whole level of the game. The player’s character, the map of cells, and the enemies present. It will run turns until the player’s character wins or dies and will draw the environment around the character after each turn is run.

**Authors:** Nihal Goindi (976005), Mariya Ahmed (990306), Tim Roger (977422).

|  |  |
| --- | --- |
| **Leaderboard** | |
| **Responsibilities** | **Collaborations** |
| Completion time.  Display and update leaderboard.  To display user information. | UserProfile |

**Class Description:** Leaderboard will display the users’ completion time and continue to update as the game goes on.  
**Authors:** Kelsey Pyne (976805), Joseff Pugh (975656)

|  |  |
| --- | --- |
| **Coloured Door** | |
| **Responsibilities** | **Collaborations** |
| Knows its own colour which will match with a KeyCard.  Opens if the player’s character has a KeyCard of the same colour.  Draws itself. | KeyCard  Character |

**Class Description:** Coloured door is a type of door which is part of a matching Key Card and door pair. It will open if the player’s character has the correct Key Card in its possession and moves onto it.

**Super Class:** *Door*.

**Authors:** Nihal Goindi (976005), Mariya Ahmed (990306), Tim Roger (977422).

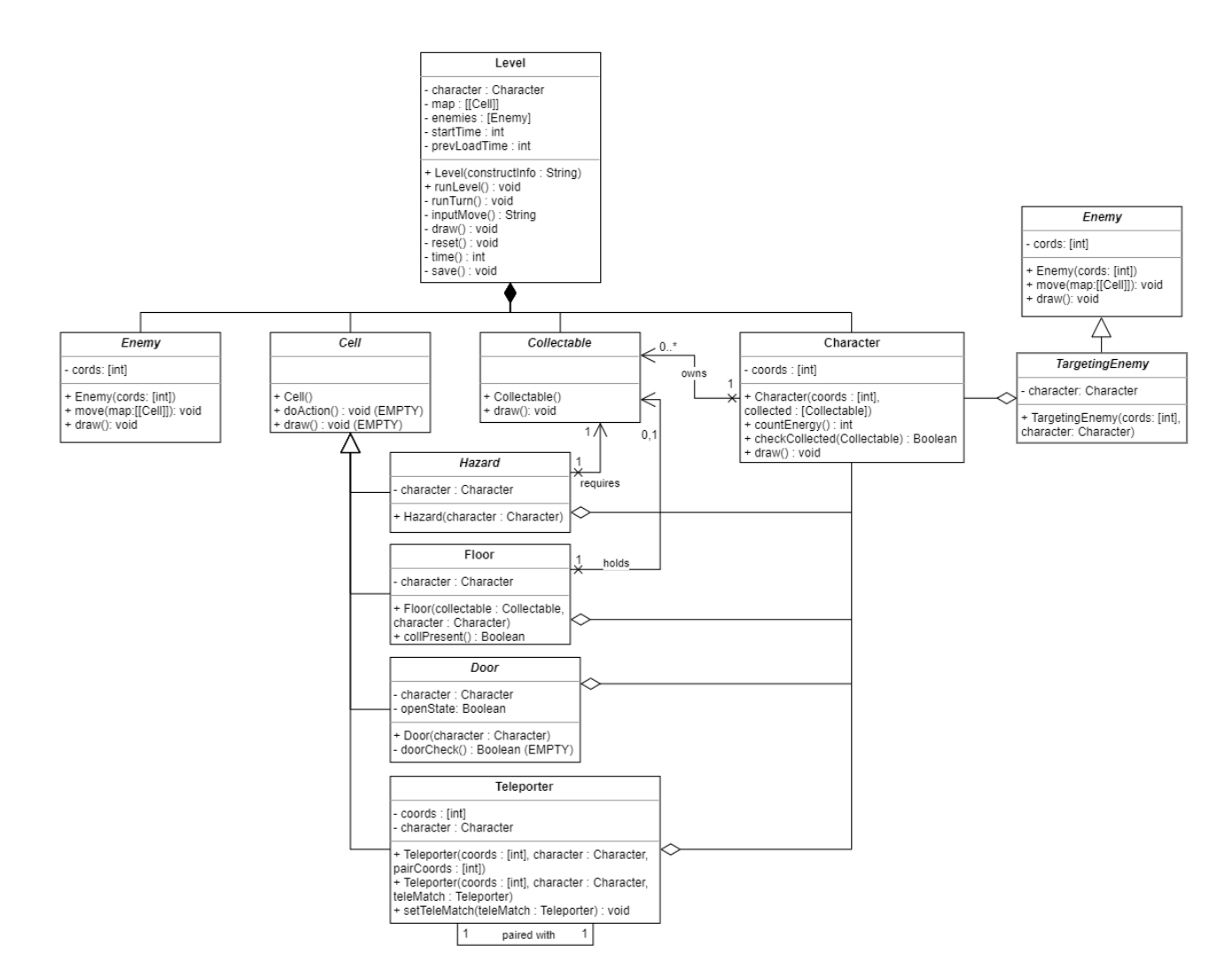
|  |  |
| --- | --- |
| **SmartTargetingEnemy** | |
| **Responsibilities** | **Collaborations** |
| A type of enemy that will follow the player well and find a way around the walls if the player is behind the wall.  A reference to the character will be passed onto the enemy so the enemy knows where to go.  Draws itself. | Cell  Character |

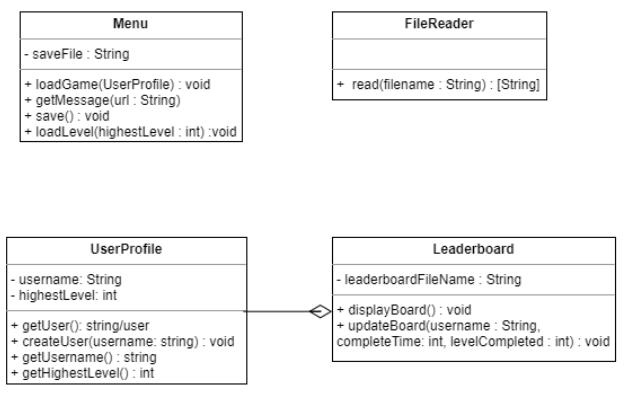
**Class Description:** This type of enemy will be smart and will walk around the walls following the player. This enemy will not be able to pass through obstacles. If it touches the player, then the player will die.

**Super Class:** TargetingEnemy

**Authors:** Szymon Grzech (988065), Jumaira Miller (983101).

**2 - Class Diagrams**





Out of all of the behaviours we have included in the UML diagram above, we believe that our 5 most complex behaviours are: Level.runTurn(), DumbTargetingEnemy.move(map : [[Cell]]), Floor.doAction(), Menu.loadGame(UserProfile) and FileReader.read(filename : String)).

Level.runTurn() is designed to run a turn when one of the arrow keys is pressed, making the players character move onto another cell, as well as the enemies within the level. As each turn is run there needs to be checks like: if the character has been caught by an enemy, if the character has stepped onto a hazard, if the character has picked up a collectable, if the player has completed the level by reaching the goal etc. Therefore this will collaborate with: Enemy, Cell, Collectable and Character as a level is composed of these components and their subclasses. There is an aggregation collaboration between targeting enemies and characters, as enemies are trying to catch the character. Level.runTurn() will also call Level.draw(), once all movement and checks are complete, in order to show the player all the information they need to make their own movement decision for the next turn.

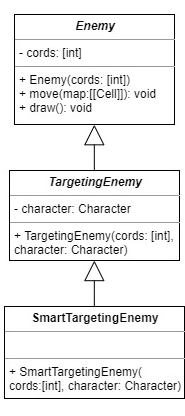
DumbTargetingEnemy.move(map : [[Cell]]) will be implemented by passing in a 2D array, all the cells (i.e. the map) in order to work out possible moves the character can make. The enemy will try to go to the characters coordinates but it is not going to find its way around an obstacle or a wall and will fundamentally work by trying to decrease the distance between itself and the character. The method will decide which direction to move in and change the coords attribute of the enemy to their new values.

Floor.doAction() works when it a player moves onto a floor cell object. It will check if there is a collectable by collaborating with its respective class and if there is one it will be removed from the cell and added to the players inventory. A Floor object can hold 0 or 1 collectable objects.

Menu.loadGame(UserProfile) is designed to load users information. Once loaded it will determine the highest level the user has achieved where they will be able to choose any lower levels and one above. It will also be able to save the level as file if the user chooses to pause the game and will be able to continue where the user previously saved the game.

FileReader.read(filename : String) is designed to read the level's text file, split it into a list of Strings and return it to the call location to be processed. It will do this using Java's File functionality.

**3 - Hierarchy Descriptions**

We have decided to design *Enemy* as an abstract superclass because all of the types of enemies will have common attributes, like coordinates on the map, for example. Additionally, all enemies will be able to move and be drawn, hence why the ‘move’ and ‘draw’ methods are designed to be empty; Subclasses of *Enemy* will have to fulfil the method move otherwise the instances will not work. Empty methods make sure that certain aspects of superclasses are followed through and that is exactly what we wanted with our *Enemy* hierarchy. *Enemy* has three subclasses, all modelling different type of possible enemy: WallFollowingEnemy, StraightLineEnemy and *TargetingEnemy*.

*TargetingEnemy* inherits everything from its superclass, *Enemy*, yet is also abstract and keeps the move method empty for its own subclasses to fulfil. This class is a generalization of any type of enemy that will have to target the Character instance’s coordinates. This class will have two separate subclasses which will implement their own way of moving inside of the move method. *TargetingEnemy* has its own reference to a Character instance because all its subclasses will need to use the Character instance’s current coordinates in relation to their own position and the move method’s map parameter. There are two *TargetingEnemies*, DumbTargetingEnemy and SmartTargetingEnemy.

SmartTargetingEnemy is a root class that can produce actual instances. Any single instance of SmartTargetingEnemy should have, because of the inheritance from its hierarchy, a list of 2 integers referring to its coordinates, a reference to the Character object and implementation of move to fulfil its own way of finding which direction to move and change its coordinates. Using its own available information and a list of Cell objects making up a map passed to the method as a parameter, the SmartTargetingEnemy object will have to use a short path algorithm to find which direction would lead it to the Character object’s coordinates the fastest.

By placing these classes into an inheritance hierarchy, we get to generalise the aspects of all the possible enemies in the game and leave room for additions to the roster. Furthermore, the efficiency of our code is increased as we will have less of it to maintain as well as bugfix in the future due to the reduction in duplicated sections. Moreover, it is essential for the abstract *Enemy* class to exist so that the runTurn method in the Level object has the ability to run through each instance of Enemy in its list and call each one’s move method without having to worry about which type of enemy it is. This should simplify the process greatly.

**4 - Level File Format**

**Part i – Format Overview**

A level file will contain at least three specifications structured within as described below:

1. Size of the map:

This will be represented as a two-number coordinate, separated with a comma (“,”) with the first number describing the width of the level, and the second describing its height.

1. Map information:

These lines (the number of which is equal to the level’s height), will map out the level using the character assigned symbols (see Part ii) to describe locations of basic cells types, such as walls, ground, player, items etc.

1. Player start point:

This next line will describe the player’s starting point upon loading the level. Note that the player’s starting coordinate may differ if they have already begun attempting the level. This line will be structured as follows:

**X,Y,0**X and Y are the coordinates of the player’s starting point.

1. Additional level information:

Any more lines added in the file will each describe additional information needed for the level and its entities, including doors, enemies, their types and other specifications needed. Each line should only describe one cell, and each cell should only have one line describing it.

Regardless of which cell they are describing, each line will be structured as follows:

**X,Y,TYPE,SUBTYPE,ADDITIONAL**

X and Y are the coordinates of this cell’s initial position.

TYPE describes the type of object that is there. The four cases this can be are DOOR, ENEMY, TELEPORTER and KEY

SUBTYPE describes the subtype of the above-mentioned TYPE.

For DOOR, this is either TOKEN or KEY.

For ENEMY, this is LINE, WALL, DUMB or SMART.

For KEY, this is BLUE, RED, GREEN or PURPLE.

For TELEPORTER, this is an integer that acts as an ID for the teleporter.

ADDITIONAL is any further information that is needed about the cell.

This part is will be NULL for KEY type cells and ENEMY types cells that are not LINE or WALL subtypes.

For a DOOR, KEY type cell, this information describes the colour of key required by the door (BLUE, RED GREEN or PURPLE).

For a DOOR, TOKEN type cell, this information describes the number of tokens required to open it (e.g. 3).

For an ENEMY, LINE type cell, this information describes the initial moving direction of the enemy (UP, DOWN, LEFT or RIGHT).

For an ENEMY, WALL type cell, this information describes the wall side it is following (RIGHT or LEFT).

For a TELEPORTER type cell, this information is the ID of another teleporter that this teleporter is paired to. A teleporter pair should only ever refer to one another in their descriptions.

**Part ii – Cell Symbols**

|  |  |  |
| --- | --- | --- |
| “#” | Wall | No character nor enemy can traverse onto or through walls. |
| “ “ | Floor | Empty space where any enemy or player may traverse onto. |
| “G” | Goal | This is the cell characters much reach in order to complete the level. |
| “A” | Acid | This cell is considered a Wall cell to enemies and will kill the character if they have not collected a Hazmat Suit (below). |
| “W” | Wires | This cell is considered a Wall cell to enemies and will kill the character if they have not collected Rubber Shoes (below). |
| “-“ | Coloured Door | This door cannot be traversed until a keycard of the corresponding colour has been collected by the player. |
| “=” | Powered Door | This door cannot be traversed until enough Energy has been collected by the character. |
| “E” | Enemy | This cell contains an enemy, if a player and enemy are ever on the same cell, the player is killed. |
| “T” | Teleporter | When a character moves onto a teleporter, they will be teleported off  to the teleporter’s pair on the opposite side. These are considered ground cells to enemies. |
| “H” | Hazmat Suit | A collectable that helps a character to traverse over Acid cells without being killed. |
| “R” | Rubber Shoes | A collectable that helps a player to traverse over Wires without being killed. |
| “K” | Keycard | A collectable keycard of a specific colour that will open Coloured Door cells of the same colour. |
| “O” | Energy | A collectable item that will open Powered Door cells when the required number of them has been collected for each requirement. |
| “,” | Empty Space | NOTHING |