Partial Design Document - Group 9

2.1.1 - Candidate classes and responsibilities:

TokenDoor – Specific class to model the token door aspect of the game – will contain information concerning the number of tokens required to open the door	
Author: Noah Stebbings Superclass: Door Subclasses: N/A	
Responsibilities	Collaborators
• Contains a more accurate draw method • Will have the number of tokens required to open the door	Player, Token

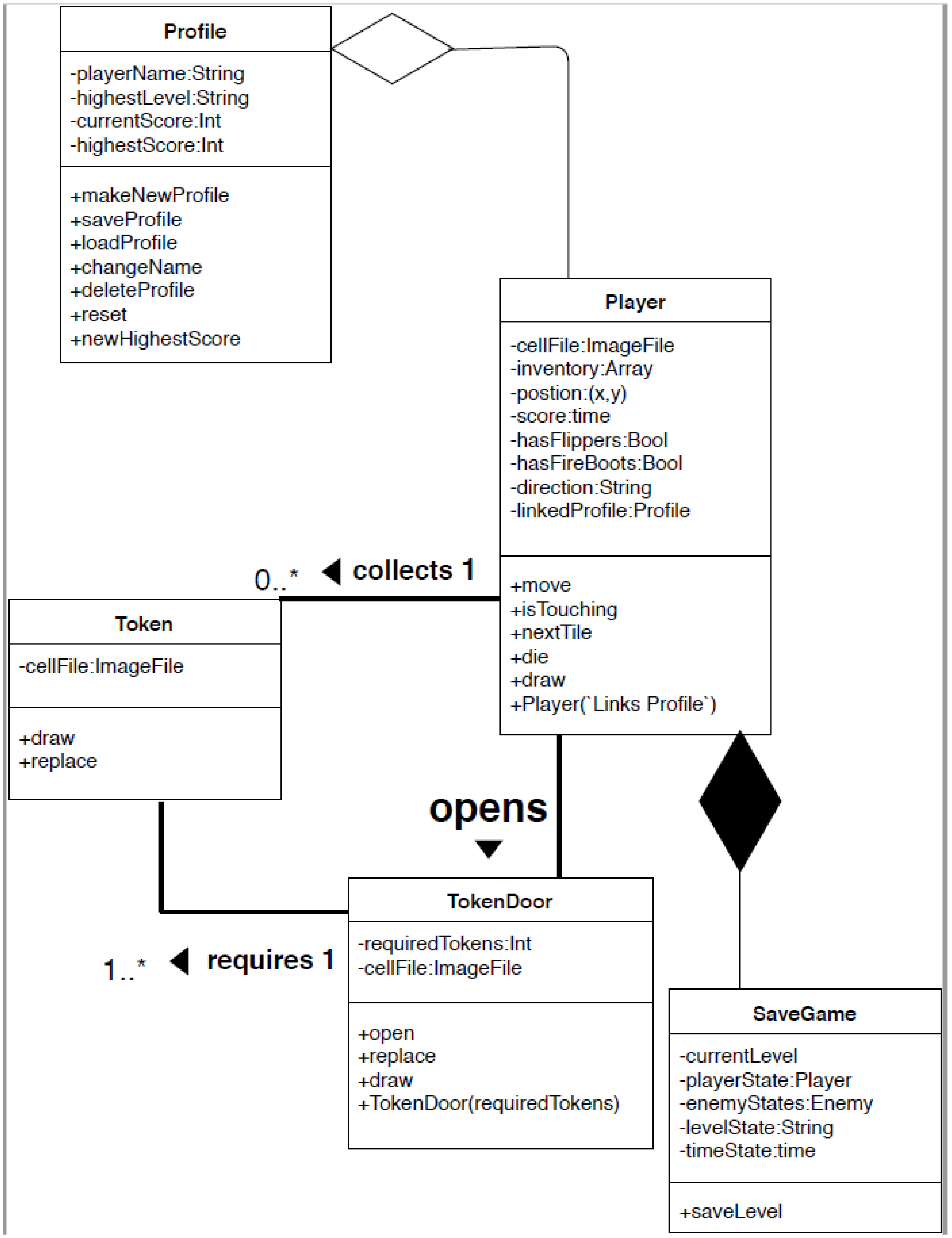
SaveGame – Designed to allow the user to save the current state of the game. The game will also be saved after every move, to ensure data persistence.	
Author: Mohammed Raihan Superclass: N/A Subclasses: N/A	
Responsibilities	Collaborators
• Will have to save current level state – such as where the player is located, what doors they've opened and enemy locations etc • Has to save the inventory of the player – such as how many tokens they have and if they have fire boots/flippers	Game, Profile

Token – Designed to house details orientating around the collectable tokens within the game	
Author: Cai Sidaway Superclass: Item Subclasses: N/A	
Responsibilities	Collaborators
• Contains a more accurate draw method • Has a 'collect' method that will call the redraw method in the **item** class, and will interact with player's inventory class, adding item to it. • Boolean *collected* will change to true, allowing enemies to walk over item	Player, Door, Item, TokenDoor

Player – Will contain information on where the player is located, and will have methods to move the player, whilst redrawing the player depending on direction. Contains information on inventory items too.	
Author: Hao Wu Superclass: Body Subclasses: N/A	
Responsibilities	Collaborators
• Contains a more accurate draw method – which will redraw the player depending on direction facing • Will contain move method to actually move the player along x-y coordinates • Will store inventory items in an array, where we will preselect array slots for each item we will pick up, for example the inventory array slot 3 will always contain the number of tokens the player possesses. It will collaborate with each of the item classes, such as token and key to fill these item slots	Body, Key, Item, Door,Profile, SavGame

Profile– This is a class that holds common information about 'existingProfile' and a 'newProfile' class	
Author: George Cook Superclass: N/A Subclasses: N/A	
Responsibilities	Collaborators
• Contains a method that creates profiles for user • Store high scores for that user • Can edit the information about the profile e.g. name. • Can also reset profile	Main, GameScreen, Player

2.1.2 - Class Diagrams:



Relationships

We have decided to use an aggregation relation on the `Player` to `Profile` because a user cannot play a game without a profile as there is then nowhere for the game to save the scores. However, the Profile and Player can exist without the other. For example, unlike having a basket that is made of rope, a `Player` is not made up of `Profiles`. We have decided to use a composition with the `SaveGame` and `Player`, you can't have a `SaveGame` without a `Player` object. It would cause an error if you tried to save a game without having a `Player` class as it a requirement for the game.

The five most complex methods:

Make new profile

This is a method that will take multiple inputs from the user and will create a new user profile for them. This will allow them to save their scores and their levels progress. Without this method no one would be able to save their progress and everyone's high score would be the same.

saveLevel

This is a method that will need to save all the details of the current level and player who is playing the game. It will effectively take a snapshot of the game, the player object and the screen layout. Translate this into a text file that can then be loaded using a load save method in the loadGame class. It will need to access the data stored in the Player object this can be achieved through getters. Which can all link to one method inside the Player class that just parses information need to the save game class. This information would be: 'hasFlippers', 'hasFireBoots', 'score', 'inventory', linkedProfile'.

isTouching

Will be a method that will detect what the current tiles that the Player is next to and whether the next move will result in a death or if it is an item that needs to be added to the inventory. This will need access to the game screen so that it knows where other cells are. It will get this information by having a collaboration with the game screen, so that it can use the positions of all the tiles and where the player will move to next. All this will need is the coordinates on the adjacent tiles from their respective classes.

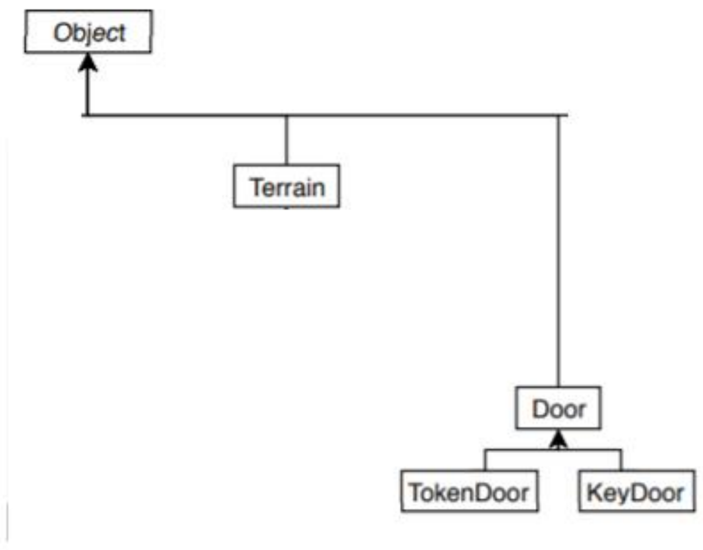
move

This method will take in user input from the user in the form of a key button press, one of the arrow keys. This will have a collaboration with the gameScreen as it must tell the graphical interface where the player should be redrawn and in turn the new position of the player character.

open

`open` is a method that will open a certain door based on requirements that need to be met, in this case how many tokens are required. This will need to collaborate with the players inventory and the gameScreen class. This is because it will have to check that the player has the required number of tokens in their inventory subtract that amount. Also, the reason it will work with the gameScreen is because it will have to call it's `replace` method and this will replace the current tokenDoor tile with a normal floor tile that can be walked over by enemies and players alike.

2.1.3 - Hierarchy Descriptions:



We have chosen to put these classes into an inheritance relationship as it allows us to use abstract methods and superclasses. We made this design choice as it removes repetition of methods and attributes, as they can be put in a more general superclass, rather than being repeated several times in smaller methods. Another advantage is that the code will be more readable, making it easier for group collaboration during the development cycle, thereby increasing the overall quality of the code. A good example of this in our classes is the *Door* superclass, and the *TokenDoor* and *KeyDoor* subclasses. Without the door superclass, both of the subclasses would require a method *openDoor()*, which would do the same thing, in opening the door when the requirements of the door are met. Using the superclass *Door* saves having to repeat this method in both classes, as it can instead just be written once inside the superclass.

In the section of our class hierarchy shown above, both *object* and *door* are subclasses of the class *object*. This class contains even more general information than *door*, such as information such as the object's coordinates, its texture, and a general draw method. *Terrain* and *Door* are not in subclasses of each other however, as they only need to share information stored in *object*, for example, both objects require *coordinates*, however *terrain* does not require a method *openDoor()*, as it is not a door, and therefore doesn’t need to be opened.

2.1.4 - Level File Format

Level File Format

Below is a basic format of how levels will be stored within their respective text files.
A file will begin with the level’s dimensions so that the scanner understands which of the next characters are of the level layout. Additional information for a level will be kept below the level layout. This includes the conditions of enemies, the requirements for doors and colours of keys.
Lastly, the file will have details on the state of the player’s inventory. This is effective in the event that the player saves a level after already picking up one or more items.

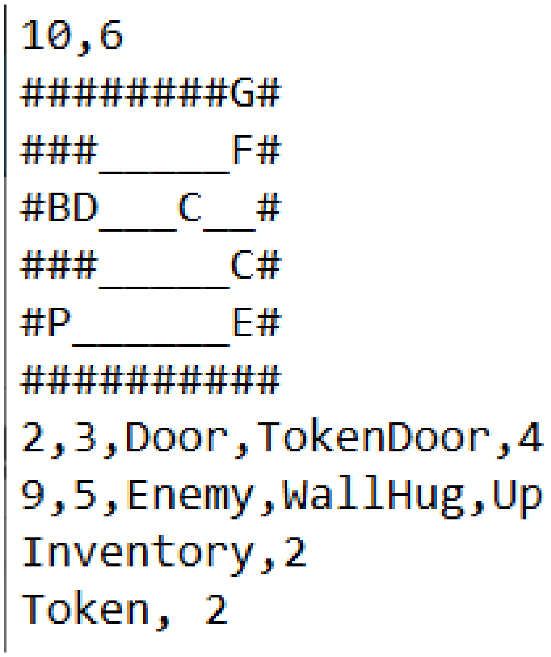
Format

1. Length, Height
2. [begin level layout]
3. …
4. …
5. [end level layout]
6. Length,Height,Enemy,Type, Direction
7. Length,Height,Door,Type, Requirement
8. Length,Height, Key,Colour
9. Inventory,NoOfItems
10. Item,Qty
11. Item,Qty

Index of character definitions

Character	Item
#	Wall
_	Floor
T	Teleporter
C	Token
K	Key
B	Boots
L	Flippers
G	Goal
F	Fire
W	Water
E	Enemy
P	Player
D	Door

Example of a stored level



```
10,6
########G#
###_____F#
#BD___C__#
###_____C#
#P______E#
##########
2,3,Door,TokenDoor,4
9,5,Enemy,WallHug,Up
Inventory,2
Token, 2
```