Partial design document – Group 9

Notes:

- Need to add actual methods like collect into our class diagrams
- There's no actual collaboration between tokendoor and token? The token door will only check the player's number of tokens
- Need a direct link between savegame and profile?
- Need to think of fifth method to write about

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

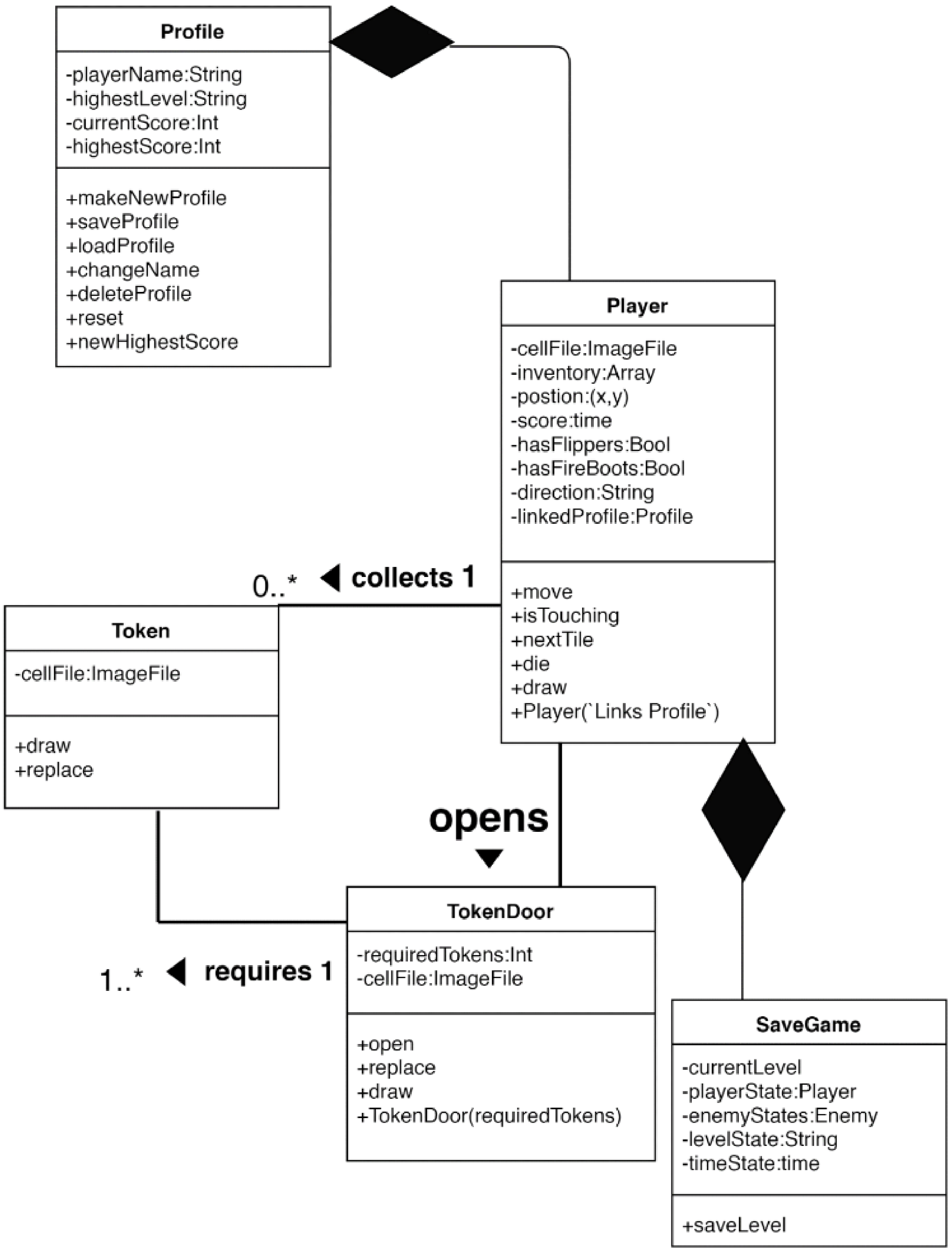
SaveGame – Designed to allow the user to save the current state of the game. The game will also be saved after every move, to ensure data persistence.	
Author: Mohammed Raihan Superclass: N/A Subclasses: N/A	
Responsibilities	Collaborators
• Will have to save current level state – such as where the player 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

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	Body, Key, Item, Door

Profile– Designed to be a superclass, holding common information about existingProfile and newProfile class.	
Author: George Cook Superclass: Body Subclasses: NewProfile, ExistingProfile	
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	Body, Key, Item, Door

2.1.2 Class diagrams



Method explanations

Collect - This method will exist in the Token class. It will add one token to the total number of the tokens that the player has and redraw the cell into a normal cell, as the token has then been collected. One player can hold any number of tokens. This can be implemented by a collaboration with the player class, as it will call the players ‘setTokenNumber’ class to equal one more than it currently does. It will then call its own redraw method to turn it into a normal cell that enemies can now walk on.

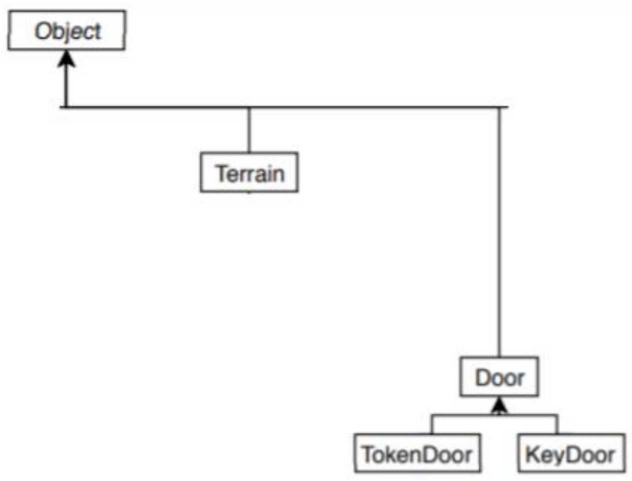
CheckNumberOfTokens - This method will be implemented in the TokenDoor class. It will check the number of tokens that the player possesses and will determine whether or not the player is able to open that particular door or not, displaying a message to the player if they’re unable to open it, or redrawing itself as a normal cell if the player is able to open it. It will be implemented using a collaboration with the player class, it will check the number of tokens the player has, and compare it to how many tokens the door requires. If the player has less – then the door will not open and display a message to the user. If the player has enough tokens, it will retract the number of tokens the player has by the number of tokens the door requires and will then open itself.

Player() – This method is implemented in the Player class – it will be used to link the player will the profile of the person actually playing the game. This will allow the Player class to parse information like the players times and current level to the profile class – so it can be saved under the name of that person and loaded later on, so progress isn’t lost. Will be implemented using a collaboration between the Player class and the Profile class, allowing information to be written to the profile of the player.

SaveLevel – This will be implemented into the SaveGame class. This will draw all necessary information from the Player class such as inventory status and player location. As not shown in the diagram it will also draw information from enemy classes, and the LevelMap class – so it can save the status of the map itself and the enemy locations. This will be implemented using a collaboration with the Profile class – so it can draw all information from the Profile class such as the name of the user

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.



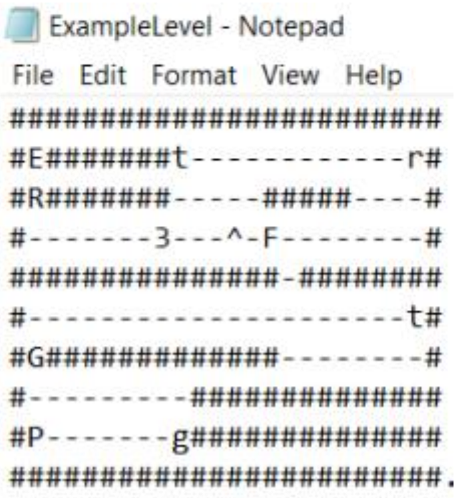
In the section of our class hierarchy shown above, both object and door are subclasses of the class object. This class contains even more general information than door, such as information such as the objects coordinates, its texture, and a general draw method. Terrain and Door are not in subclasses 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

This is an example of how we plan to create the levels. We will use a basic ASCII format in order to hold information about what tile is in what location.

In this example:

- "#" is a wall tile
- "E" is the end goal
- "t" is a token
- "-" is a floor tile that can be walked on
- "r" is a red key
- "R" is a red door
- "3" is a token door requiring 3 keys to open
- "^" is a basic enemy that moves up and down
- "F" is a fire tile that can only be walked over by a player with fire boots
- "G" is a green door
- "P" is the player start location
- "g" is a green key
- "." is a symbol that tells the code that there is no more level information after this point.



```
########################
#E#######t------------r#
#R#######-----#####----#
#-------3---^-F--------#
###############-########
#---------------------t#
#G#############--------#
#---------##############
#P-------g##############
########################.
```

A file reader would read through this document when the level is loaded and would be able to translate this into information that can then be used to generate the levels in the game.

The document would then be followed by additional information, such as the direction certain enemies move, etc. This information would just be stored as a series of integers and Strings, that would make no sense on their own, but the file reader would be able to interpret.