Group 1

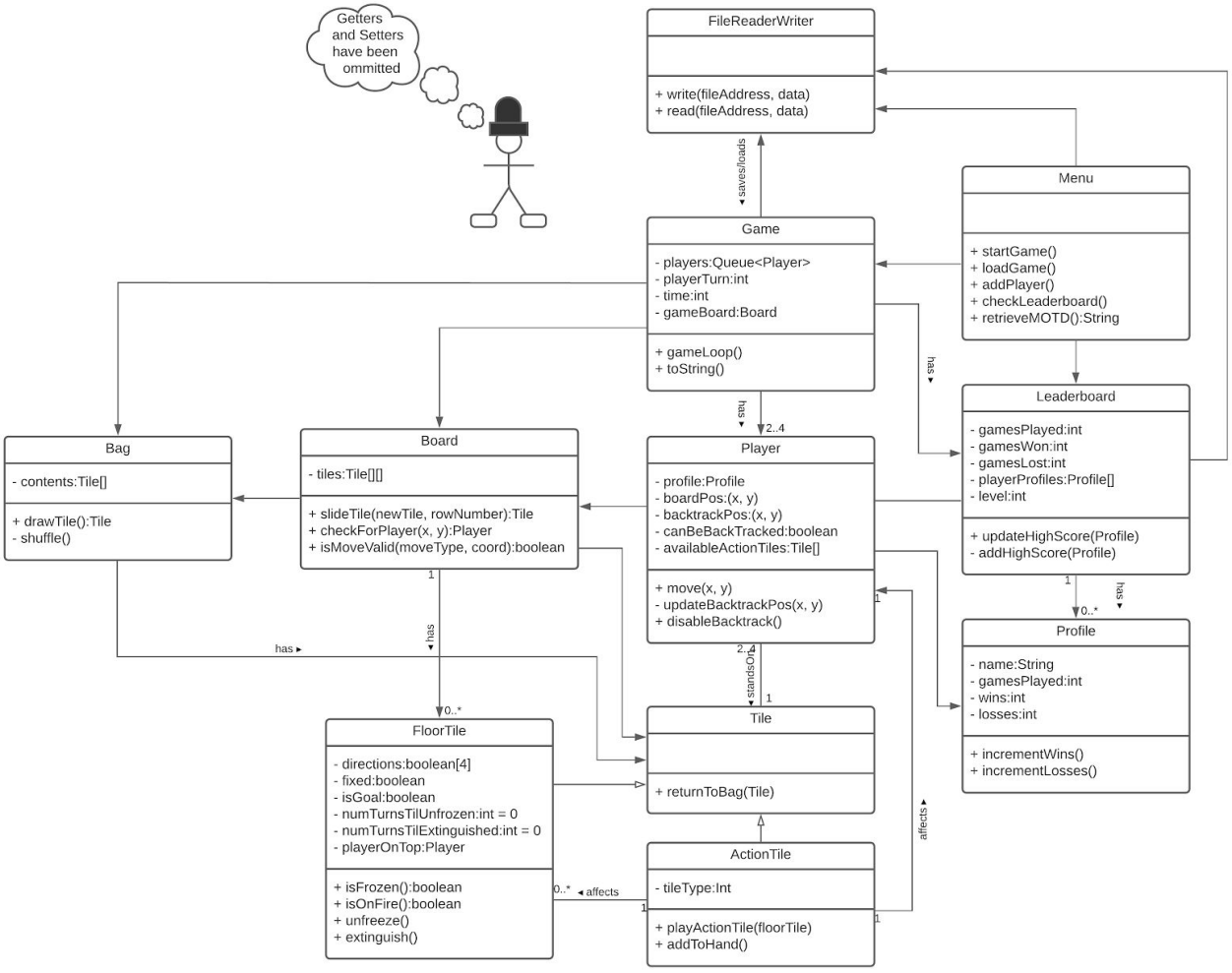
Partial Design Document

2.1.1 - CRC cards

Class	**Description**	**Author (et al.)**	**SuperClasses and Subclasses**	**Responsibility**	**Collaborates with**
Profile	The profile will hold the permanent data for an individual player, including wins and losses on each individual level, as well as a player name.	Lukas	N/A	A profile will be selectable before the game starts, and will be tied to an instance of a Player class in the game. Once the game is completed, the Player class will pass if the player won or lost, and on which level the game happened. This may include extra stats, such as number of tiles played, distance moved etc.	Player, Leaderboard
Player	Represents the person playing the game, including their character's location on the board.	Lukas	N/A	Link to the player's profile Track the player's position on the board Track the position to which the player should backtrack if required. Store all action tiles that the player possesses.	Board, Profile, Leaderboard
Leaderboard	Displays a list of the profiles and their overall stats. The list will be ordered based on the stats of the profiles, going from the profile with the most wins downwards.	Jacob	N/A	Each leaderboard will be tied to an individual level, and will refer to each profile that has at least 1 game played on that level. It will then order them based on win/loss rate. This will also be saved permanently along with the profiles.	Game, Menu, Profile, FileReaderWriter
Game	Runs and controls the various entities in the game.	Tim	N/A	Track number of turns Track current player Track time Store game board	Player, Board, Bag, Leaderboard, FileReaderWriter (to load and save the game state).
Menu	Holds functions to be used on the menu outside of the game loop	Glen	N/A	Allows the players to control the program and its available functions. Menu will give the users the ability to save and load games, add new players to the database, check the leaderboard, and access	Leaderboard Game FileReaderWriter

				the MOTD	

2.1.2 - Class Diagram



5 most complex methods

Name	Description	Means
startGame()	Initiates the game	It asks the players to input their names. It then creates Player instances, constructs a new Game (passing in the players), and creates a FileReaderWriter. Then it calls gameLoop on the Game object.
gameLoop()	Runs the main game (disregarding menus etc)	The next Player in the players queue is removed. The drawTile method is then called from the Bag object. The tile is then either placed or kept depending on its type. If it is a floor tile then the slideTile method is used from the Board object to slide the tile onto the board. The player then can choose to play an action tile from their availableActionTiles array. The Player

		position is then moved using the move method from the Player object. The Player object is then moved to the back of the players queue.
slideTile()	How the player manipulates the board with floor tiles	slideTile will be called after a player has taken a floor tile from the bag, with the newly collected tile and the desired row/column passed to it. The player will choose either a valid row or column to slide the tile in. Every other row and every other column is a valid entry point. The function will also check that there are no frozen tiles in that row/column (which makes that row/column invalid). It then moves that tile into the correct position on the board, and slides the existing tiles. This will then call the return to bag function on the last tile, and also move any player pieces on that tile onto the new tile.
write()	Saves the game to a new file	Converts all the data for the current game into text format in a .txt file that allows it to be read again by the reader. Must do this for both the leaderboard and the current game, must also be able to update existing files with new data whilst keeping parts of the initial data intact.
checkLeaderboard()	Displays the leaderboard	Creates a FileReaderWriter and uses it to read data from the leaderboard file. Creates a Leaderboard and populates it with the data. Displays it.

2.1.3 - Inheritance description

The sole inheritance relationship in our design is that connecting Tile, FloorTile, and ActionTIle. ActionTile and FloorTile are both subclasses of Tile. Both action tiles and floor tiles need to be stored together in the Bag object as the player is required to draw one at random from the bag, however, Java uses homogeneous lists, meaning we would not be able to store action tiles with floor tiles. Storing the action tiles and floor tiles separately would not be a very elegant solution as we would need to balance the probabilities of drawing them. However, seeing a thing in common between action tiles and floor tiles - the need for both to be drawn from the bag - we chose to generalize the idea of a Tile via an inheritance hierarchy.

2.1.4 - Save/Level file format

Description

Board is an x by y grid stored at the top of the save.Bag contains n pieces
Each tile must have:
- it's movable directions (Up,Right,Down,Left)
- if it is a goal
- if it is a player spawn (0 for no, 1-4 for that player)
- turns until the tile is unfrozen (always 0 for levels, but can be more for saved games)
- turns until the tile is extinguished (")
- if it is an action tile (Ice,Fire,Double,Back; 0 for no, 1-4 for each type, all other attributes must be false or 0)
`true,true,true,true,false,0,0,1,0` would be a 'plus' tile that is not a goal, is not a

player spawn and is not an action tile that isn't frozen but is on fire for 1 more turn.
`true,false,true,false,false,1,0,0,0` would be a straight path going up/down that is not a goal, spawns player 1 and is not an action tile and isn't frozen or on fire.
`false,false,false,false,false,0,1,0,0` would be an action tile (ice).
The tiles are in a list starting with the x*y board tiles, then the rest are bag tiles. None of the first x*y tiles can be action tiles.
The goal can only go on a non-movable row. Non movable tiles must be in an even position in the array (counting from 0) and who's position must be even (ignoring decimals) when divided by x.
Players are spawned in on the board's default spawn points. When saving an existing game, the player's current location is stored as the new spawn point.
However, the players name, backtrack position, available action tiles and if they can be backtracked must be stored as such
(1) // indicates number of players in save, if 0 then it is a new game and menu selection values are used
`1,Paul,(4,6),True,True,4:1,2,2,3`
This notates in order, the player number, profile name, backtrack location, if they can be backtracked, if it is their turn (player 1 by default) and the number of action tiles they have, and which ones they are.

<u>Example save game for a 2x2, 3 player game</u>

```
2x2
(3)
1,Paul,(1,2),False,False,4:1,2,2,3
2,Jeff,(2,2),True,False,0:
3,Wayne69,(1,1),True,True,2:4,4
False,true,true,true,false,3,0,0,0
True,false,false,true,false,1,0,0,0
True,true,false,true,true,2,1,0,0
True,true,true,true,false,0,0,0,0
True,true,true,true,false,0,0,0,0
True,true,false,true,false,0,0,0,0
False,true,false,true,false,0,0,0,0
False,false,false,false,false,0,0,0,3
True,true,false,true,false,0,0,0,0
False,true,false,true,false,0,0,0,0
false,false,false,false,false,0,0,0,1
```

(end of board)