Semestral work

detailed specification

Marek Sádovský
MFF UK

Contents

Introduction

The goal of the project with the name Azul is to create both, a command line and a graphical version of a turn-based color tile drafting game based on Azul.

This document represents the specification of the resulting work from the point of view of both user and developer functionalities. It determines what is to be done by the developer and how it should behave from the user's point of view. A detailed description of the program implementation of the goals presented here will be provided after the finalization of the project by the Technical Document. However, the major part of it is mentioned here as well. The User documentation will present instructions for controlling the supplied SW and game rules.

Chapter 1

Use cases

1.1 Graphical

The young talented kid who has been refused from art school decides to show that his impressive artwork is worth seeing. So he decides to call his only two friends and shows them this computer game called Azul where they in 3 clicks start a game and taking turns start to build impressive mosaic work. They all can see each other process, and control everything using the mouse. Each turn the game calculates their score for them so they don't need to use hard math formulas.

1.2 CLI

A few guys from Matfyz are bored because it's summer and they already finished all their assignments. They start to look for some game and they find the Azul. Because of their love for their hundred-year-old ThinkPad, they are happy to find out that this game has a command line version so they can play it even in their beloved terminal, using intuitive commands.

Chapter 2

Game description

Introduced by the Moors, azulejos (originally white and blue ceramic tiles) were fully embraced by the Portuguese when their king Manuel I, on a visit to the Alhambra palace in Southern Spain, was mesmerized by the stunning beauty of the Moorish decorative tiles. The king, awestruck by the interior beauty of the Alhambra, immediately ordered that his own palace in Portugal be decorated with similar wall tiles. As a tile-laying artist, you have been challenged to embellish the walls of the Royal Palace of Evora.

The setup of the Azul game is composed of the **player's board (from now on board)** one for each player, **factory displays (from now plates)**, and a **center** where there are sometimes dropped tiles from plates. There are tiles of multiple types and one unique tile which determines who will start the next turn (also referred to as **brick**, if you wonder why try to go shopping with brick and I guarantee you that everybody will let you through). The board is made of some more things. You can find the **wall(grid)** there, that is the place you want to build your mosaic. To be able to push something to the grid you need to fill a **buffer** which has a unique length for each line at the grid. On the board, you can also find a **floor**, which is placed on board you won't like. Even the best of ancient mosaic artists had problem that sometime when they worked they dropped something and then they needed to clean that up. So for each tile on the floor, you would be given a penalty in the form of negative points. The last thing on board is the player's score, which determines the winner.

There are two main parts of the game. In the first part, players take turns in choosing from which plate they take tiles. Each player each turn can choose to take all tiles of one type from any plate or from the center (if he is first to take from the center he also gets a brick, which goes on the floor because it's unsafe to use it on wall). If the player chooses to take tiles from the plate, everything else on that plate (what he didn't take, it falls) goes to the center. If the player chooses to take from the center other tiles stay there. After taking the tiles he needs to decide on which buffer he wants to put them in, he can only choose one buffer where there can't be another type of tile. Chosen buffer is than filled by tiles in hand. If a player finds himself having on the floor more than seven items he won't get more tiles on the floor instead they leave the game immediately. When the player has done all this, the player on the right side continues with the same options.

When all plates and the center are empty(so players can't choose anything) game moves to the second part. In this part player's actions are independent of each other. Everybody starting from the buffer on top if the buffer is filled takes one tile from it and places it in the corresponding row on the grid, rest of the tiles from the buffer are destroyed. He calculates how many points he gets. After going through all buffers, the player clears his floor and gets so many negative points if he has the brick he will be starting the next turn. After this, we need to check if the game isn't finished, and if not the dungeon master puts new tiles on plates and the game is on.

2.1 Parameters

Plate - the number of plates is determined by the number of players times two plus one

- for 2 players 5 plates
- for 3 players 7 plates
- for 4 players 9 plates

For the first part of the game on each plate are put 4 tiles of random choice

The game is composed of **100 tiles** in a total of 5 types (20 tiles each)

The player's grid is of size 5 times 5 where each row and each column can contain only one tile of each type

And buffers are sized as follows

- first row of the grid is length 1
- second row of the grid is length 2
- third row of the grid is length 3
- fourth row of the grid is length 4
- fifth row of the grid is length 5

The floor's size is 7 tiles, if more tiles should be put there they are destroyed instead

In calculating the points for filling to the grid we give one point for the tile put there now and plus one for each in the row without empty space between and the same for the column where the tile is. After the game, there are special points to gather for each fully filled column (7pts) or row (2pts) and for placing the maximum allowed number of the same type of tile (10pts)

2.2 Move

There are two parts of the game

2.2.1 Factory offer

In this part starting with the player who had brick as last (first round choose randomly). The player can choose to take all tiles of one type from any plate or from the center (if he is first to take from the center he also gets a brick, which goes on the floor because it's unsafe to use it on wall). If the player chooses to take tiles from the plate, everything else on that plate (what he didn't take, it falls) goes to the center. If the player chooses to take from the center other tiles stay there.

After taking the tiles he needs to place them. His options are to put them in some buffer or drop them on the floor(which brings negative points and isn't preferred unless necessary). He can choose the buffer based on one of these options

- the buffer is empty and the corresponding row in the grid doesn't contain this type of tile
- the buffer contains this type of tile but isn't full yet

If he chooses the buffer whose size is smaller than the number of tiles he collected this turn, the rest of the tiles (those that don't fit into the buffer) go on the floor. After this, the player on this player's right can go.

2.2.2 Wall filling

When all the plates and the center are empty, the game enters the "Wall filling" part. in which each player evaluates his board as follows, going from top to bottom throughout the buffers.

- if the buffer is full, we take a tile from it and place it on the grid in the corresponding row and the rest of the tiles in the buffer are destroyed
- otherwise we ignore the buffer

There are two ways to play the game, basic and advanced. In the basic game, the grid has predetermined where in the row each tile type is supposed to go. In the advanced one you can freely choose where to put it but there can't be two or more tiles of the same type in one row or column.

After putting the tile in the grid you receive points for it. If it's placed such that there aren't any other tiles directly above, below, or to the left or right, then immediately gain 1 point. If there is a direct next to it (left and right) count the number of touching each other tiles in that row. And if there is a directly above or below count also the number of touching each other tiles in that column.

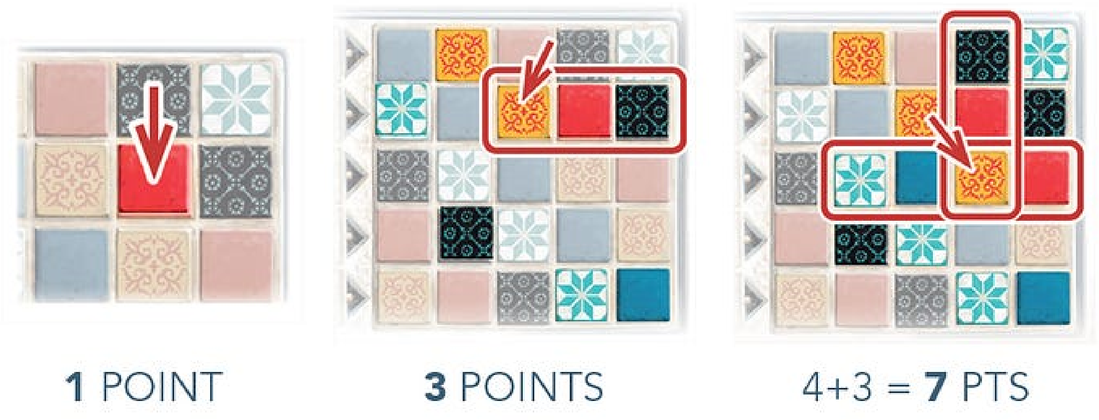


Figure 2.1: Points example

Chapter 3

Games views

In this chapter, we take a closer look at how the windows of the game should look in graphical version and also how the prompt will be formatted, what is where, and what means what.

3.1 GUI

In the graphical interface, everything is set up for mouse control, ensuring smooth navigation throughout. The only button used is **esc** to exit the game which can be used at any time, the game then asks you if you are sure, and after authorization it terminates. In the following part you can see every window preview (it's just a prototype)

3.1.1 Menu

In this view, we can see the name of the game and the buttons providing the following operations

- **Play** - starts the game with the number of players selected lower
- **About** - this will provide the player with the rules of the game and how to control it
- **Credits** - this will show the player credits of the game

Between the Play button and the About button, we can see a section that sets how many players will play. By clicking on plus or minus we can add or remove players (maximum of 4 and minimum of 2 players). For each player, you can write the name under which he will be seen in the game.

3.1.2 Game

In this view, players hang out most of the time. Everything happens here. They can see their board, as well as the **boards of other players**, along with their **scores**. Plus, they get a good look at all the plates and the center.

Since the game is on one device, when the player finishes his actions, a simple screen is shown with the next player's name and the button which "moves" us to his view. He can play. This happens in both the first and second phase.

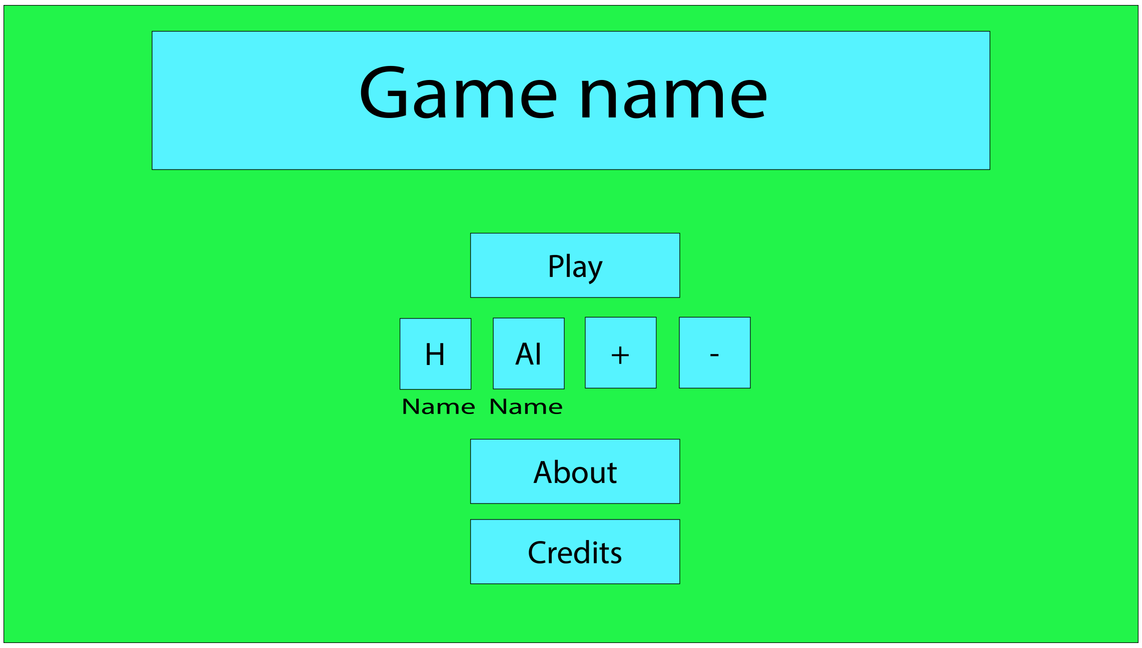


Figure 3.1: Menu

Board

In the 3.3 Board view, we find a detailed board. It consists of the following parts

- **Grid** - a wall where the tiles are put from buffers
- **Buffers** - represents holders that which player tries to fill to be able to add tile to the grid
- **Floor** - this is the place where tiles that weren't fit to buffers are put
- **Score** - shows how many points a player has.

Table

Here, 3.2 we have a comprehensive view of the entire game, as mentioned earlier. During a player's turn, they have the ability to click on a tile that is situated on any of the plates or in the center. If we click on the chosen tiles, corresponding tiles will move with our cursor, until we choose a buffer or the floor and click on it. This will only work if he can add the tile there, otherwise, a message will be shown and tiles will be returned.

In the second stage of the game, when players place tiles from buffers on the wall their board is zoomed in so they can see it better. Then the buffers from top to bottom are highlighted, and also the row in the grid. Based on the version, they can choose the position by clicking on it.

On the right side we can see the other players boards, which are the same as the current players board, along side with the name of the player. We can see plates with 3 different colors, red ones are in every game, orange are if there are at least 3 players and yellow are only in 4 player variant.

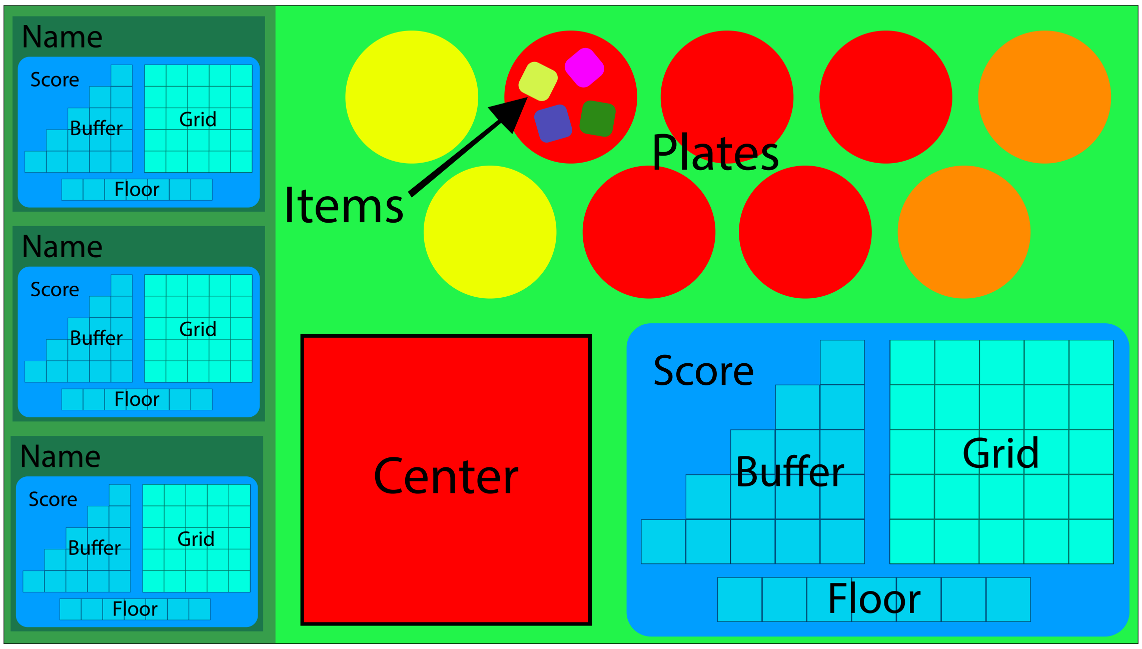


Figure 3.2: In game view

NOT TODO:*when hovering by the cursor on another player's board it will zoom on it same with the own board plates or center. We can grab the tiles and drop them to buffer*

3.1.3 Game over

After the end of the game, 3.4 End game panel will be shown with scores of all the players with the button which will allow us to go to the menu.

3.2 CLI

This part will explain what the command line application will look like.

When the game starts it asks you how many players are playing and after getting a valid answer it generates the game. Which will look as follows:

- Others - here is the list of enemies (other players) with their board in row-ordered data. In other words, the row on top is written first in the format of buffer, and the grid row second is next, and so on. After all the rows of the grid, there is the floor. Each element is separated by two spaces, with each player occupying one line.
- Table represents the plates and the center, and the number of columns corresponds to the number of the plates plus the center. Rows represent the type, so in every position we can see which type of tile is there and from where we are taking it.
- In me section we have our board displayed similarly to the others.
- action represents the place where there is a cursor waiting for our command. A valid command will tell where we are taking it from, and where we want

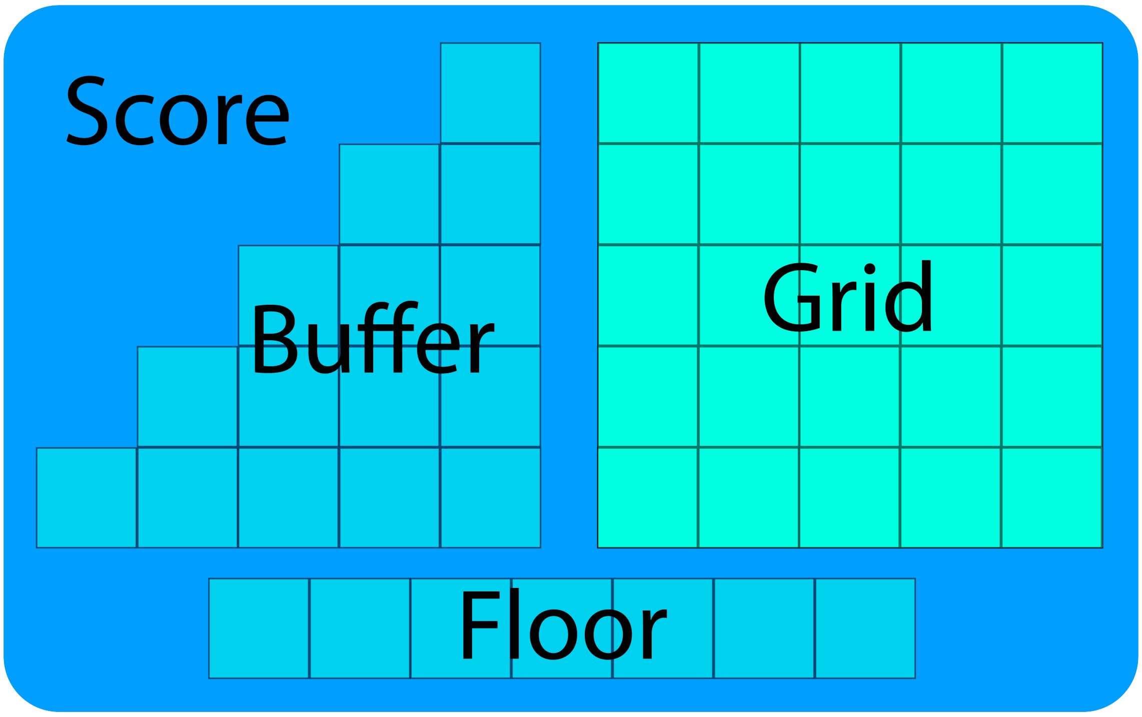


Figure 3.3: Player's board

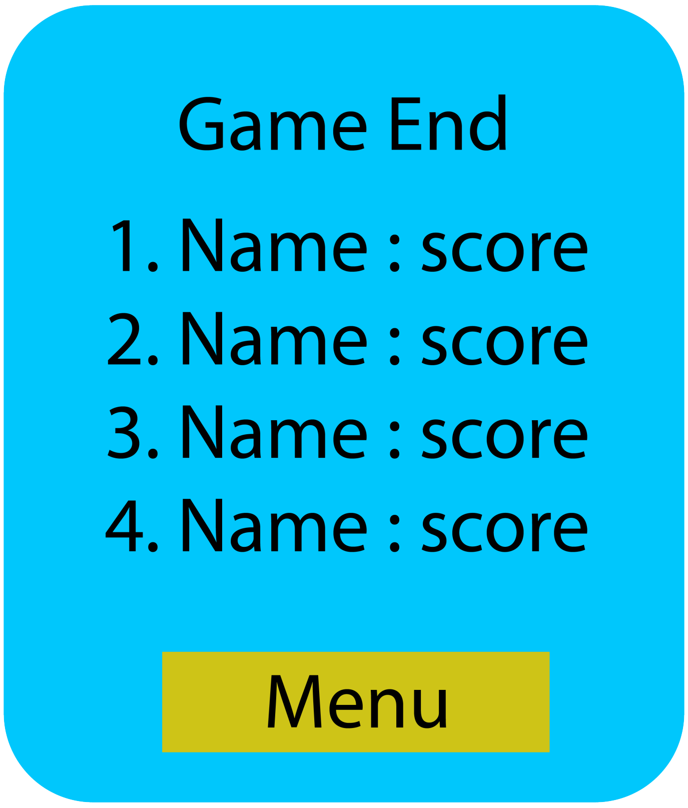
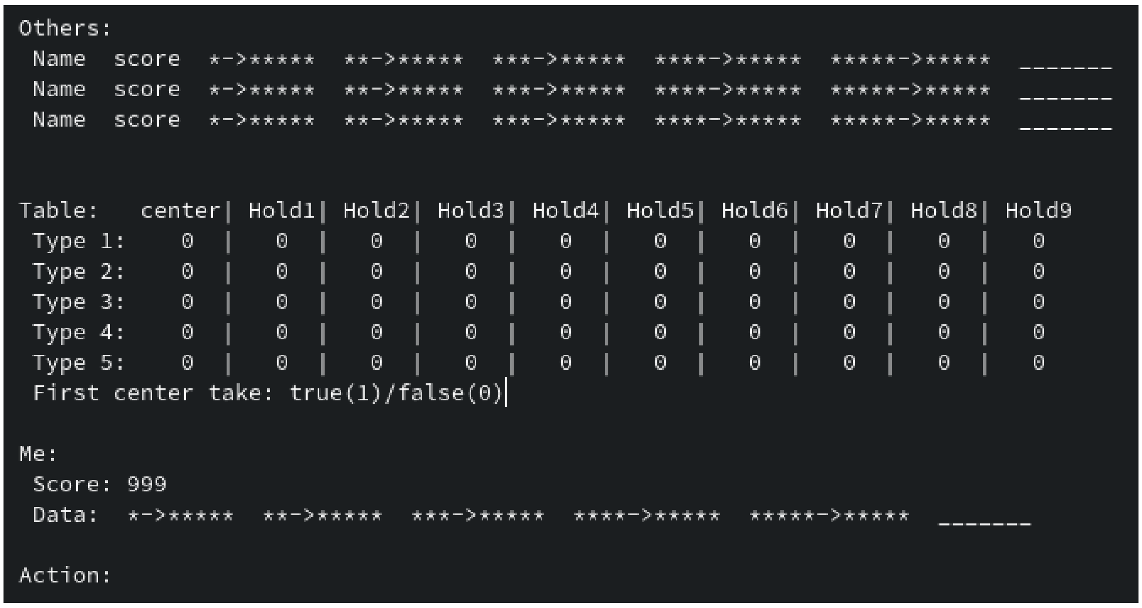


Figure 3.4: End game panel



```
Others:
 Name  score  *->*****  **->*****  ***->*****  ****->*****  *****->*****  _______
 Name  score  *->*****  **->*****  ***->*****  ****->*****  *****->*****  _______
 Name  score  *->*****  **->*****  ***->*****  ****->*****  *****->*****  _______


Table:   center| Hold1| Hold2| Hold3| Hold4| Hold5| Hold6| Hold7| Hold8| Hold9
 Type 1:    0   |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0
 Type 2:    0   |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0
 Type 3:    0   |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0
 Type 4:    0   |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0
 Type 5:    0   |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0  |   0
 First center take: true(1)/false(0)

Me:
 Score: 999
 Data:  *->*****  **->*****  ***->*****  ****->*****  *****->*****  _______

Action:
```

Figure 3.5: CLI in game

to place it. If we are in the faze one `<C or H1-9> <1-5> <1-5>` which is in format from where, what and where to put it. In the second faze game tell us which row (in CLI is more of the column) we are on so we know what we are placing and we only write position `<1-5>`. For example: H3 3 4 - takes all items of type 3 from the hold3 and put it into the 4-th buffer.

After the game is finished the result will be shown similarly to the GUI but directly in the command line. The application will be closed.

Chapter 4

Technical specification

In earlier chapters, we covered the user interface and the game rules familiar to players. Now, let's delve into the intricate technical details of how these elements will be translated into practical implementation within the system. To start take a look at the basics:

- Supported platforms: Linux, Windows
- Language: C# with .net 8
- Libraries: Unity libraries (Unity, UnityEngine, etc.)

4.1 Structure of the application

This application is composed of two parts:

- I/O handler: this both in graphical and command line handles what the player sees, and controls his inputs
- Game logic handler: here all the logic will happen, this part contains the game cycle, realization of the rules, and checking win condition.

The input/output handler for the graphical interface will be implemented using Unity. In contrast, the CLI version is designed to run independently of Unity. This entails that the graphical version will feature a distinct GameManager compared to the command line version, and will also incorporate additional scripts to manage the output specific to the graphical interface. Meanwhile, the game logic stays the same no matter the interface we choose, which means that the code will be the same for both options. So in this part, we look at the logic implementation, and then at the I/O.

4.2 Game logic handler

Before diving into the implementation of the game logic, we should take a look at how player's data are stored and handled.

4.2.1 Player

In this structure, we will store the data of one player, and some methods that will check possibilities, win conditions, and handle storing or loading the data.

Properties:

- `string name` only for user experience
- `int pointCount` (from outside only getter)
- `int[,] wall` (from outside only getter)
- `buffer[] buffers` (these will have a method to get their values, buffer type is specified in the helping structures.
- `Tiles floor` representing the floor
- `bool isFirst` checking if will start next round

Methods:

- `bool Place(int row, Tile tile)` if it's possible it will automatically add it to the chosen buffer, and rest to the floor then it returns true, if it's impossible (wrong type or illegal buffer) it will return false.
- `Tile getBufferData(int row)` returns the specific buffer from buffers
- `int[] fullBuffers()` returns indexes of the buffers that are full and can be placed on the wall
- `bool Fill(int row, int column)` tries to fill in the wall type from the buffer at a specific column, if it's illegal, it returns false, otherwise it returns true, this method can trigger an event at the end of the game.
- `bool ClearFloor()` this clears the floor and changes point-count and returns if is first

4.2.2 Tile

This represents the structure where are stored data of the tile

Properties:

- `int id` this stores the type of the tile
- `int count` this stores count of the same type at some place

4.2.3 Tiles

This structure helps to operate with separate tiles.

Properties:

- `int[] counts` for the i-th index (id) stores count of it and it's getter from outside

Methods:

- `public Tiles(int types, int count)` this is a public constructor where we specify the number of types, and the count is the number of tiles of the specific type
- `private Tiles(int[] counts)` private constructor to generate Tiles with specific counts
- `Tiles GetRandom(int k)` randomly choose k elements, removes them from parent Tiles, and generates a new Tiles object with the chosen elements
- `int CountOfTiles(int id)` returns counts[id]
- `int TotalTiles()` returns the number of all tiles in structure
- `Tile GetTiles(int id)` removes and returns the number of tiles with specified id
- `void PutTile(int id, int count)` add count to specific index in counts
- `void Union(Tiles other)` add other to own values

4.2.4 Plate

This represents the structure of the plate on the table.
Properties:

- `Tiles tiles` representing the tiles on this plate
- `bool isEmpty` If it's true then no operations are allowed here (publicly only getter)

Methods:

- `Tile[] GetCounts` this returns the non empty elements of tiles
- `void PutTiles(Tiles tiles)` this stores the tiles and remove previous data from tiles
- `Tile TakeTile(int type)` takes the Tiles with the id == type returns Tile and set it to zero in tiles

CenterPlate

It's a special kind of the Plate which inherits all the methods and properties and has extra ones
Properties:

- `bool isFirst` this represents if anybody took something from the center this turn

Methods:

- `void PutTiles(Tiles toPut)` add Tiles to existing ones

4.2.5 Board

In this class, we handle the main logic of the game. Also here we handle if it's a basic or advanced mode, so in basic mode, there is also an empty Calculate which will do everything automatically. This method is the only access point to the game data that may be used in way of continuing the game that started previously (save/load mechanics).

Properties:

- `Player[] players` this refers to specific player's data
- `Plate[] plates` this refers to the Plate structure from where we can take the tiles
- `CenterPlate center` special plate for center
- `int[,] predefinedWall` for basic mode, predefined wall
- `Tiles storage` is generated on construct and contains all the elements on the
- `int currentPlayer` helps to see whose turn it is
- `Vector2 calculating` on x is currently calculated player and on the y is the row we are on
- `int phase` to recognize if we are in phase 1 or two
- `bool isadvanced` to recognize the game type

Methods:

- `bool Move(int plateId, int tileId, int bufferId)` specifying from what we are taking where true is returned if it's okay and false if something doesn't work. If all plates (center included) are empty, change phase to 2 (Move now throws IllegalOption exception)
- `bool Calculate(int column)` this will take calculating and returns if it's a legal position or not, in every iteration we also add the points (in original we do it on end but still in the order of the addition so it's the same), if it is the last row and legal we calculate floor to and clear it
- `onWinCondition` calculates bonuses of players, also finishes the Calculate() and returns the tuple array in the format Name, score

In the move and calculate methods we are also creating a log. The logs will be stored in a unique file specific to the game, they will be in string format and can be found in a later specified directory. We create messages for every attempt of a move (even if it's illegal) and also for the calculate method. In the following format `<player name and ID> <takes/places> <from> <to> <success/fail>` As it is in txt format for string per line, we can simply filter things using tools such as grep. Later on it can be used to load the game or replay it. This class also implements getter methods to get data from Player and Plate which are securely separated from I/O handling.

4.3 I/O handling

In this part, we will have two implementations, **GUIGameManager** and **CLIGameManager** based on the version we are trying to implement. Both classes will communicate with our game and also players, so it will write and read everything from the console. After running it players will be asked how many players are going to play the game. After the legal answer (2 to 4 names), the game will start. Constructing the board will be handled in the construct so this manager only asks for data from the board and returns move() over and over till the second phase starts. At this stage, we will still be asking for data and will be handling the filling to the wall. Also, the board has an event that triggers the end of the game, so we implement it (the event sends names and scores).

Differences between the classes are specified in the visual part and the rest of the implementation of these classes is based on them.

4.4 GUI I/O handling

The game, based on its complexity, will contain 2 scenes, one for the menu and things connected to it. The second one will be for the game itself.

4.4.1 GUIGameManager

This class will implement what should be when shown, it will internally run the game, and based on the game state it will display everything needed. It will also send data to specific unity game objects that will display them in response (e.g. For the board instance, we gen who's turn it is, then it activates that player's board).

4.4.2 ButtonHandler

This class will provide functions for clicking buttons, it will load content, start the game, handle the adding or removing the players.

4.5 Bot logic

This will be one class which, when spoken to, will ask for data from the board calculates posibilities, and randomly choose one. This way when it's bots turn, the game manager calls a method on this class. Methods will be the same as in the board script, Move will have parameters from the board and return parameters for the Move in the board, and the Calculate will have which row we are filling and returns column.

Chapter 5

Not to do

In the end I would like to write out some things which won't be covered in this version. This version is not prior to creating smooth visual effects nor the animations, although some of them will be there. There won't be any smart implementation of artificial intelligence or bots, only pseudo random acting bots for testing. The game won't have any progressive graphics or sound effects. This version also doesn't require save/load mechanics which will definitely be added later.

List of Figures