Maze Game Design Document

By Rory Gallie

# 1. Purpose

With the specification given this document will outline the design process, thoughts and features to be implemented within the maze game for Olde Worlde Phunne

# 2. Scope

The scope of this document is limited to an overview of the design of a “maze game”, detailing design decisions, features and user interface. The design will fall inline with the requirements given in the specification brief “*BCS Digital Industries Apprenticeship Software Development Technician Project B - Maze Game Version 1.2”* The deliverables are to be design information such as features, user interface UX and overview of the design perspective for this “maze game”

# 3. Goals and vision

The goal of this project is to have a working and tested maze game that is fully functional including some design given from the requirements brief.

The goal of this document is to outline the design of the “maze game” that is required for implementation and development of code.

# 4. Requirements

The requirements for this design are as follows (taken from requirements brief):  
  
A Maze has a set of any number of Rooms.

• Rooms are connected by means of Passages.

• Each Room may have between 1 and 4 Passages leaving it, in the directions North,

East, South, or West.

• Each Passage with isExit == false will connect two Rooms, which might be the

same.

• Such Passages are bi-directional, i.e. will take you back to where you came from.

• Each Passage with isExit == true will connect to exactly one Room.

• There can be only one Room with an exit Passage.

• Each Room contains a set of Items, where an Item is either a Treasure or a Threat,

and has a name.

• Each Treasure has an Integer value, e.g. Gold (value 100).

• Each Threat may be defeated by one Action, e.g. a Troll may be defeated by the

action called Club.

• Once a Treasure has been collected or a Threat defeated, it will disappear from the

Room for the remainder of the game.

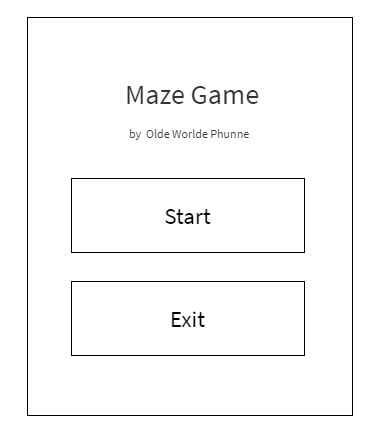
• A player must be placed at a random location within the maze that is not a wall or passage.

•When the player takes the exit passage, play finishes.

# 5. User Interface design

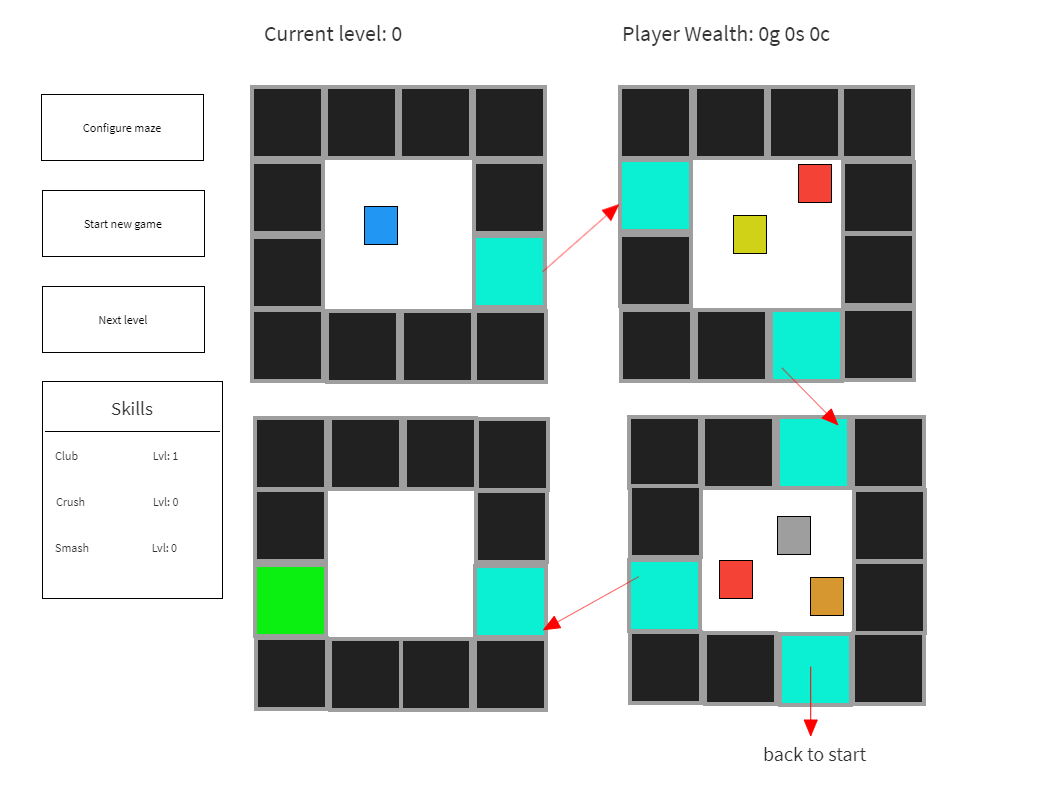
For the user interface this will be done using windows forms in C#.

The following design is for the initial window once starting the executable:



The image above shows the start-up image, this is to start the game and will initialise the main game screen. This can be used for future purpose of adding user logins to this separate form.

The main Game screen is as follows:



This shows the initial outline of the maze and information relevant to the player.

The buttons on the left hand side are to refresh the maze with “configure maze”

Start a new game which will reset to the first level.

Next level button will go to the next level if an exit has been reached.

Skills are for defeating threats with 3 skills in the game these will be eventually displayed with purple icons that will have a name of which skill it is and what level it gives.

The following icon for blue is the player location, Red icons signal a threat, gold/silver/copper are the wealth currency for the game.

The teal areas are passages to the next room. (passages could require keys and should be implemented once an initial version has been done and any changes should be added to this document)

At the top is a display for the current level and player wealth.

Each room will be loaded as a player enters the “Passage” for that room.

# 6. Data management.

The data within this game functionally will be the following:

* Configuration file for maps in the terms of a binary text file. A 1 being a wall, 2,3,4,5 being a passage for each north, west, south and east location. 6 being an exit block.
* The game will use various data types within the code relevant to the use e.g. int, float, double etc.
* Random locations will use a seed.
* Data should be managed to link everything into one main class file which will allow for most of the main functions for the form.

For the format specification of the user interface the following will apply:

* All data displayed on the screen will be done within the form class.
* The form class will access the main code by inheritance to access member variables e.g. getting the current wealth value of a player or obtaining coins data.

# 7. Functional overview

The code design will have to be dynamic to keep the system consistent and allow for an easy implementation of new features.

7.1 Inheritance and polymorphism

With inheritance and polymorphism, the best use of these is to create a base block. This base block will then be polymorphed into other blocks such as walls, passages, threats and currency.

Allowing for the easy implementation of a new block simply by polymorphing the current block.

The initial block will also have a polymorphed block for a grid system. This grid will display over every block in the maze and will allow for clear display of player movement.

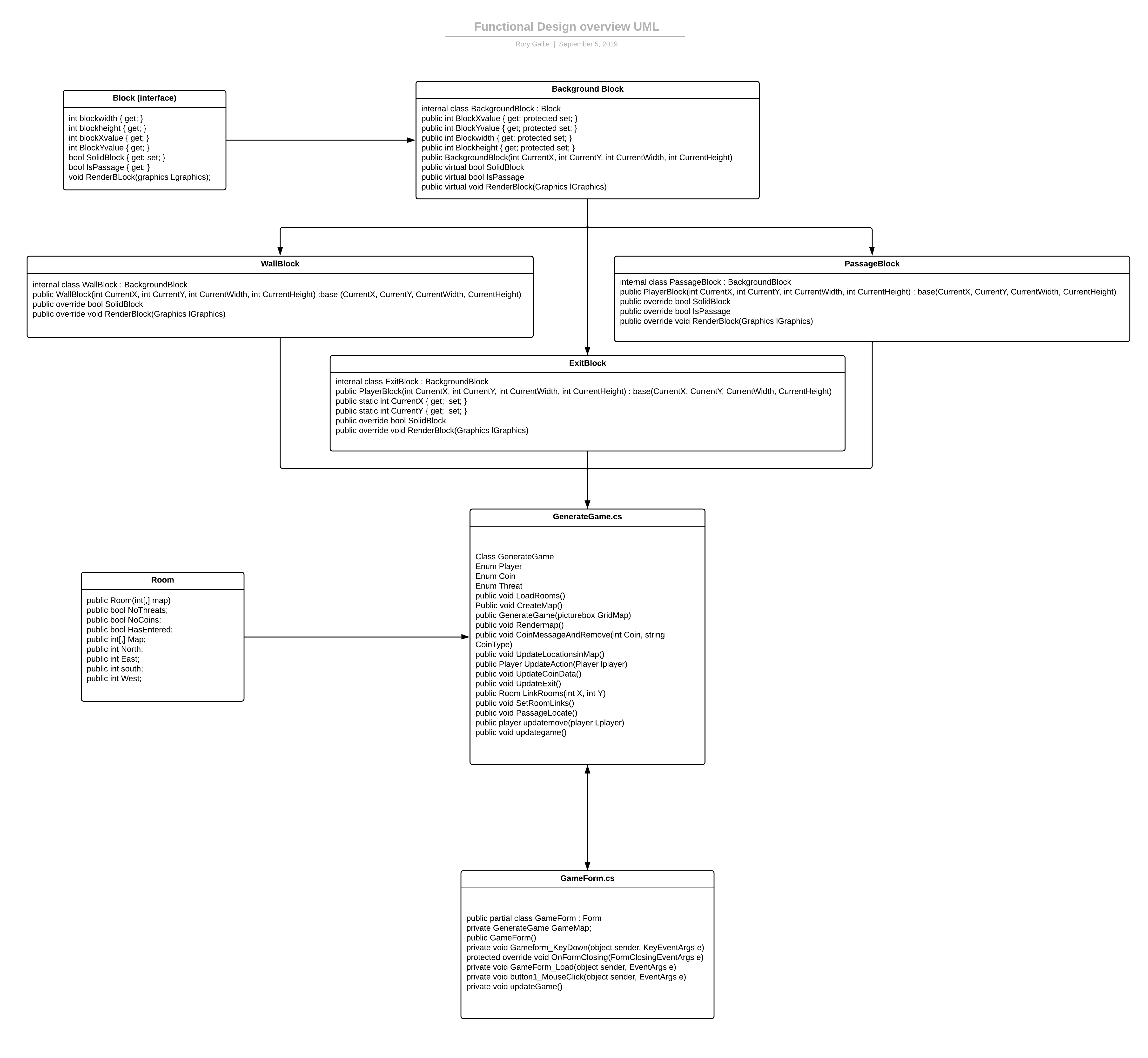
7.2 functional UML

The above UML is an initial design for the functions within each CS this will display what the functions should be name and what the design looks like from an inheritance class system.

I have pre-set what would be needed for the program to run this is from a functional stand point and does contain the basis for each .cs file.

This allows for quick and easy access to add more blocks as different types that will link into GenerateGame.cs

This is just a brief overview and changes should be done for a more dynamic system as needed. (any changes should be logged in this document)



8.0 Design changes

With this current design the Main Game form has been changed from the user interface standpoint

The new user interface is as follows: