Roll Number: ---

Assignment Title: Assignment: A JavaFX app utilising Design Patterns

Contents

[Introduction 2](#_Toc102083910)

[Overview of my game 2](#_Toc102083911)

[Provided Video 2](#_Toc102083912)

[UML Diagrams 2](#_Toc102083913)

[Class Diagram 3](#_Toc102083914)

[Use Cases 5](#_Toc102083915)

[Design Patterns & Key Features 6](#_Toc102083916)

[Singleton design pattern 6](#_Toc102083917)

[Immutable design pattern 6](#_Toc102083918)

[Utility classes 6](#_Toc102083919)

[Main centralised controller 7](#_Toc102083920)

[Conclusion 7](#_Toc102083921)

# Introduction

## Overview of my game

For this assignment, I opted to create an Automation game, loosely based off Factorio and Satisfactory. The premise of this game is you start off with raw resources, such as iron ore, and create a network of conveyors and buildings to refine and process the materials into more luxurious items. In turn, these items can be further processed. Commodities can be sold for financial gain, which may be reinvested into more buildings and conveyors.

My game is intended to indirectly teach young children the ability to problem solve. This is a key skill in all sciences, especially Mathematics and Physics. Playing the game requires getting the correct resources from place A to place B, requiring users to solve said problem. A direct connection will most likely be inefficient, requiring conveyors to be split and merged to create the optimum input to each building, which requires planning beforehand.

All game artwork was created by myself. No third-party items were used. It is suggested that Java 18, the latest stable release of Java, is used to run the provided Jar file.

## Provided Video

The video that accompanies this written document is split into four parts:

* The beginning of the game, showing the functionality and UI.
* Further into the game, where the user has built a larger factory.
* A worst-case scenario, where approximately 7,000 objects go from being stationary to moving.
* A more realistic scenario, where the 7,000 objects incrementally begin moving.

The performance shown my differ on different devices. For the final two parts of the video, there are two alternations made to the normal game: I had provided a near infinite amount of money, so I could place all of the conveyors, and the title of the app was changed from “Automation Game” to show the current fps. This should have no noticeable effect on performance.

The games centralised controller runs using an AnimationTimer. Please note that an AnimationTimer attempts to match the refresh rate of the user’s primary monitor, and as such a 60hz monitor will not see more than 60 fps. All movement scales with the fps, to provide a consistent experience.

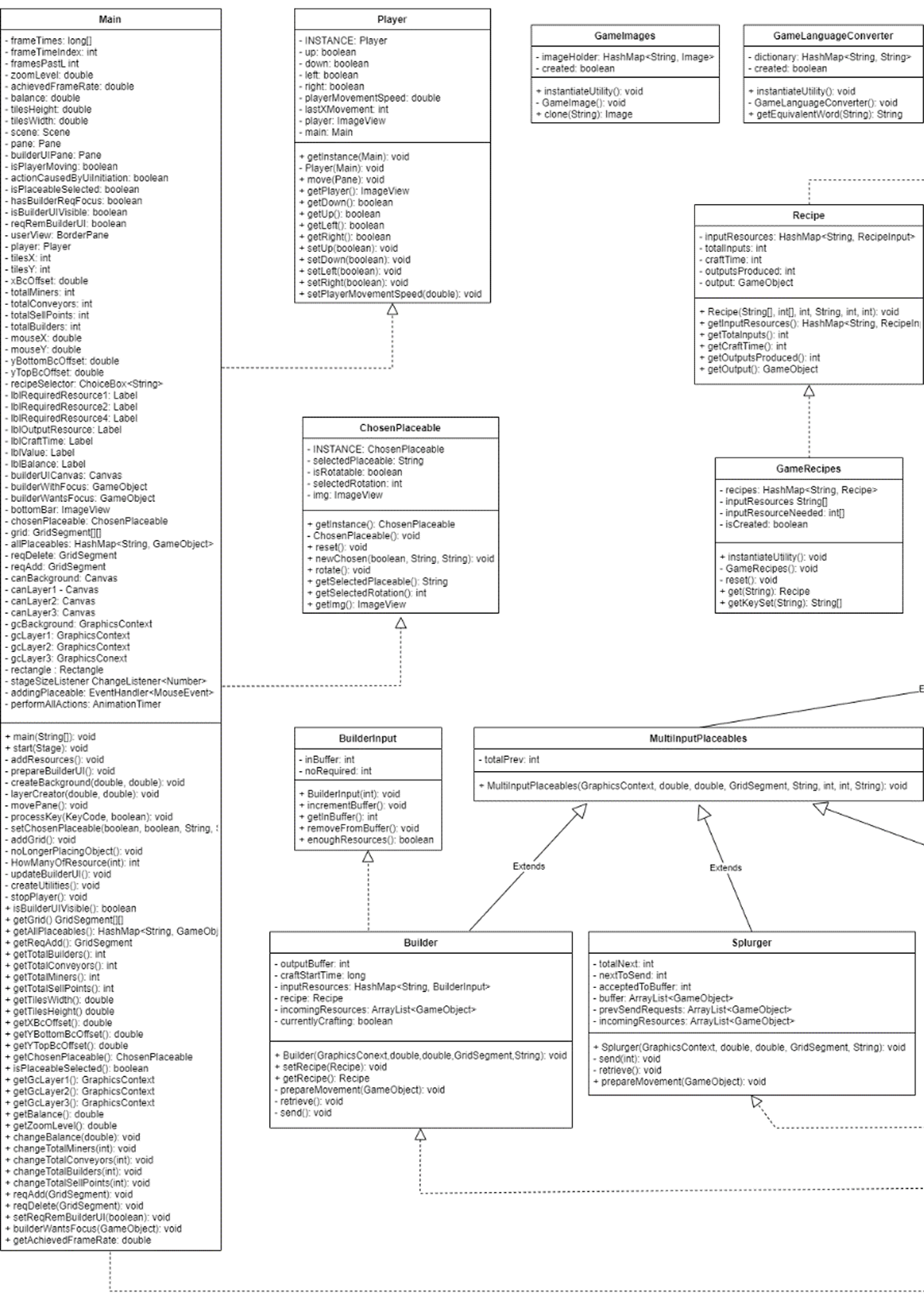
If there are any issues accessing the video, a copy of it has been uploaded to YouTube, and may be accessed [here](https://www.youtube.com/watch?v=BehmumTV3aY). The video is set as ‘unlisted’ and as such, the provided link is the only way to access said video.

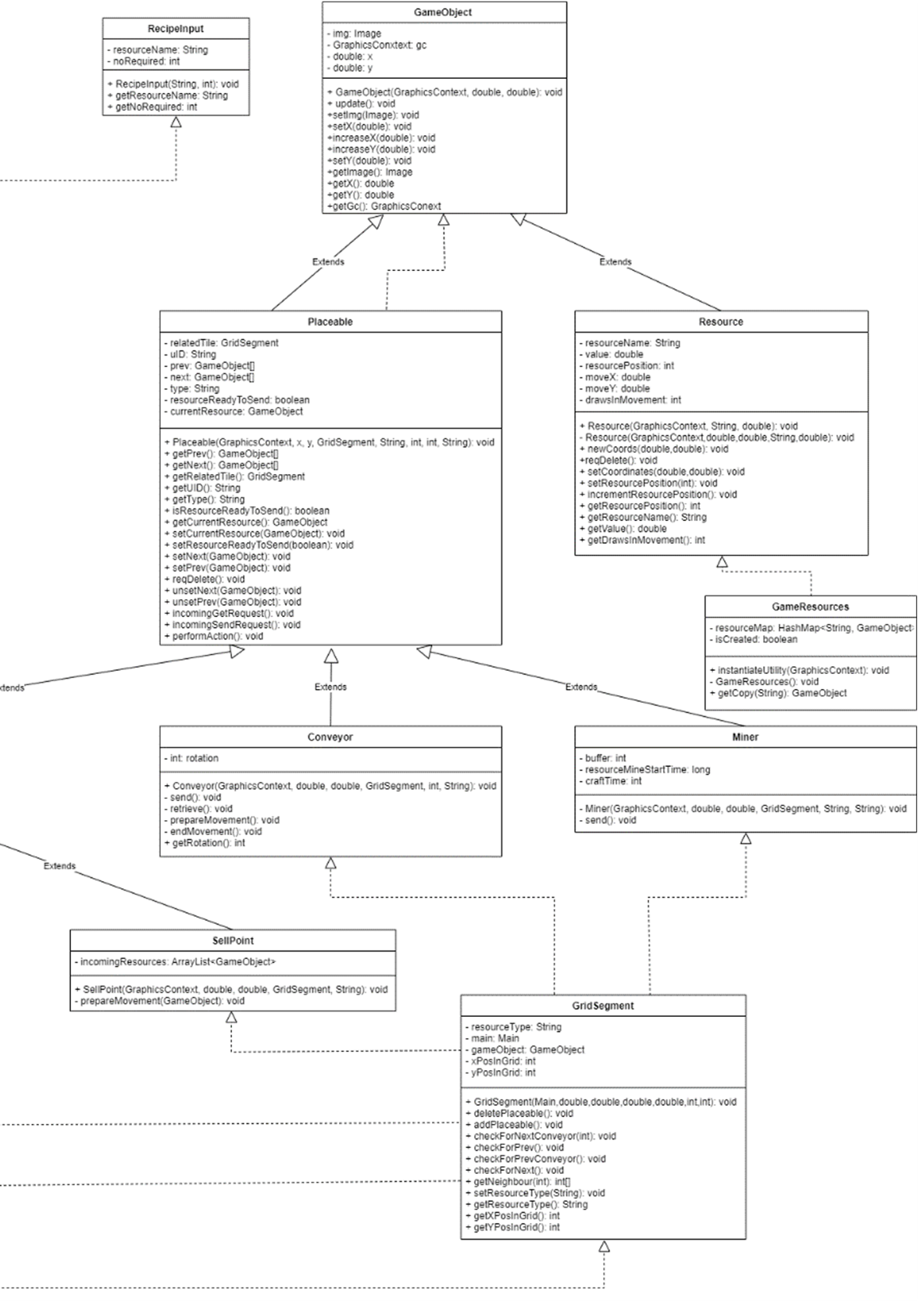
# UML Diagrams

Due to the complexity of my system, and the very limited page count for the written part of this assignment, I have chosen to approach this section as the following:

* I have not created a sequence diagram since many actions can occur many times in any order. The diagram would either be very large and not fit on this document, as well as complexed/hard to read, or would consists of many sequence diagrams on an action-by-action basics, which also wouldn’t fit on this assignment.
* Some Use Cases have been provided. This covers a small percentage of potential user operations. All operations could not be covered in this document, due to the restriction of the page limit.
* A Class Diagram has been provided, detailing all inheritance and implementation (that being where classes instantiate other classes). All dependencies could not be fully represented in a readable format, especially in the case of classes such as GameImages, that operates as a utility and is used by most classes. Representing all dependencies, would require a separate Class Diagram for every class to be readable, which the set page limit of this assignment does not allow.

## Class Diagram





## Use Cases

|  |  |
| --- | --- |
| Use Case Title | Placing Miner |
| Primary Actor | User |
| Goal | To place a resource miner on the map |
| Scope | The Game |
| Preconditions | User has opened the game |
| Postconditions | Resource miner is placed on the map |
| Main Success Scenario | 1. User pressed key “3” to select the miner placeable.  2. User moves mouse pointer to a valid tile (one with a resource node)  3. User clicks valid tile  4. User presses another key to deselect the miner placeable. |
| Extensions | 1a. User presses the wrong button – wrong/no placeable selected.  2a. User selects an invalid tile – Placeable will not be placed.  2b. There is no valid tiles left – User cannot place miner.  3a. Valid tile is already taken – User cannot place miner.  3b. User cannot afford miner.  i. User cannot place miner if there is already a miner on the selected resource node.  ii. User can place miner for free if there isn’t a miner on the map.  3c. User doesn’t click tile – Nothing happens.  4a. User doesn’t press another key – Program will indefinitely wait for user to place another miner until user presses a key. |

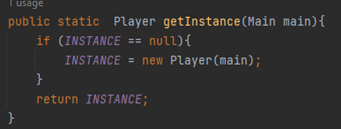
|  |  |
| --- | --- |
| Use Case Title | Moving User |
| Primary Actor | User |
| Goal | Move around the map |
| Scope | The Game |
| Preconditions | User has opened the game |
| Postconditions | Player has moved to a new location |
| Main Success Scenario | 1. User presses key that signifies movement  2. User releases key that signifies movement |
| Extensions | 1a. User presses wrong key – Nothing happens.  1b. User presses multiple movement keys – User moves in composite direction.  1c. User moves player past view of the camera –Camera moves, so user can see their player.  1d. User tries to move player out of bounds – Teleport the player to the other side of map  2a. Player doesn’t release key – Player moves indefinitely. |

|  |  |
| --- | --- |
| Use Case Title | Deleting a building/placeable |
| Primary Actor | User |
| Goal | Delete a building/placeable |
| Scope | The Game |
| Preconditions | User has opened the game |
| Postconditions | Building/placeable is deleted |
| Main Success Scenario | 1. User presses ‘Del’ key  2. User hovers over placeable they intend to delete  3. User clicks placeable. |
| Extensions | 1a. User presses wrong key – Nothing happens.  2a. User hovers over empty tile – Nothing happens  3a. User clicks wrong building – It is deleted |

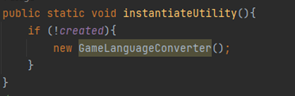
# Design Patterns & Key Features

## Singleton design pattern

Singletons were used in multiple cases in my game. A singleton prevents more than one instance of a class from being instantiated, by creating a private constructor and static methods, accessors and mutators. This design pattern was ideal for the Player class, as only one player can exist, and all utility classes (GameImages, GameLanguageConverter, GameRecipes, GameResources), as creating more of each of these would provide no extra functionality and would be detrimental to performance.



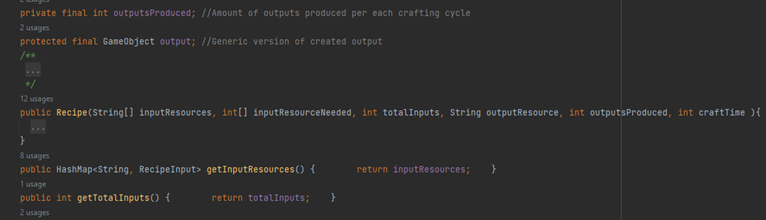
The screenshot above is from the Player class. As it shows, to get the only instance Player, one must use getInstance. This returns the Player, if one is created, or creates a Player and returns it.



Utilities work in a similar way, except they do not return the instance to whom called it. This is because all methods and variables are static, and can therefore be accessed without a pointer to the class (GameImages.clone()).

## Immutable design pattern

All utilities and Recipe classes are designed with the Immuatable design pattern. This is where all global variables are private and there are no mutator methods available in said class, preventing the change of values for said variables. Where appropriate, these variables are also declared as “final”, to prevent their value from changing. Instead, all variable values are declared in the constructor, and accessor methods are provided to access all global variables.



This is a snippet from Recipe. As it shows, all variables are private, with only accessor methods. This allows for an instance of Recipe to be shared between multiple classes via pointers, saving resources and removing further creation time of identical instances of this class.

## Utility classes

My game, as mentioned above, contains four classes that act as Utilities. These instantiate other classes, or commonly used items, so they may be accessed from everywhere, using methods such as clone() and get().

For example, the Recipe class is immutable, and may be required many times by different Builders. Therefore, rather than creating a new instance each time, GameRecipes can be used to create a pointer to the already instantiated Recipe, saving time and resources.

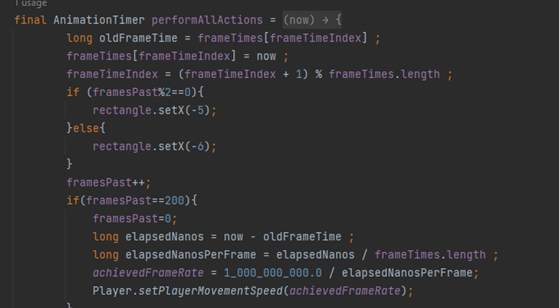
Text

Description automatically generated

Equally, all images used in the game are added to a HashMap in GameImages. Whenever an image is required, GameImages is used to create a pointer to where that image is in ram. As 1000’s of images can be required at once for my game (many of which are the same image), this utility helps to reduce how much memory they require (as everything that requires the same image points to the one copy of it in ram), and eliminates the very slow process of fetching images from the storage device.

The technique used for this is, or at least very similar to, the Prototype design pattern, and is also used in the Resource class.

## Main centralised controller



A key part of my game is the centralised controller (the above image is a small part of this). This controls the order everything happens in each frame. This ensures that the JavaFX Application Thread, which Java uses for all graphical operations, isn’t overloaded. I originally didn’t have this, which resulted in a very unstable system when 1000’s of items was attempting to draw to the screen at any given time. This controller allocates CPU time for every Placeable, to allow them to do what they require, as well as other functions, such as moving the player, and displaying/removing the UI for the builder.

# Conclusion

I believe what I have accomplished with this assignment was great, given the limited time restrictions. This was relatively complexed to make, and I believe my attempt is efficient and utilises some design patterns very well. I have also reused code where possible, and provided comments throughout my code, making it easier to read and debug.

My application also provides a consistent conveyor/player movement speed, regardless of what the fps is (including fps drops (movement speed updates every 100 frames)). The part of the game the user sees (a Pane that acts as a camera) also moves with the player, and the game resizes with the window. I am very pleased with these features.

However, there are certainly things that I wish were done better. For example, I went into this with limited planning and not being entirely sure how to create it/if I even could. This resulted in myself thinking of better solutions at later points and writing sections of code/the whole thing again. There are sections since then that I have thought of a better solution for but didn’t have the time to rewrite.

All placeable buildings inherit from the same superclass: Placeable (which is a subclass to GameObject). Every building is created by GridSegments, which needs to know exactly which building they are creating (such as Miner or Conveyor). I believe this would have been better handled using the Factory pattern, which I did not implement.

The main controller, which is mentioned above, currently handles all operations that need to occur each frame. Ideally, this should only be used for operations that require the thread for graphical changes, with everything else being handled separately. However, this not the case. Despite this, my game still performs very well, always maintaining 60 fps+, but it would be more efficient to split this.

My main class is very large (779 lines of code), and it may be more beneficial to be split this into separate classes. This would be my main focus, if I had more time, to improve readability.

Currently, there is only one building with the function of turning resources into more ludicrous resources, which makes it harder to balance the game. I did this, due to lack of time to create more game images and implement the functionality of said buildings. Given more time, I would add a Smelter, and split the builder into three buildings, in charge of crafting resources which require one, two and three inputs respectively. I would also add more resources to the game.