Multitasking Game

Report

Pietro Olivi
Leonardo Tassinari
Lorenzo Dalmonte

April 4, 2023

Contents

Chapter 1

Analysis

The software aims to create an application designed to enhance psycho-motor skills through a gaming experience based on multitasking.
The term *Multitasking* refers to the ability of a person or a product to do more than one thing at a time.

1.1 Requirements

Functional

- Upon starting, the software will display a simple minigame.[1]
- After a short amount of time a new minigame will appear and so on until all four minigames are shown.
- Generally, the difficulty of each minigame shall increase over time.
- The player's goal is to survive as long as possible. After failing any minigame the application will display the final result, therefore the software has to keep track of how long the player has lasted in the current run in order to calculate the score.
- It will be possible to appreciate the improvement on a *Statistics* page that will show the record of all the past runs.

[1] A minigame is a potentially-never-ending challenge that requires simple actions from the player in order to keep the game going.

Non functional

- The application shall sustain acceptable frame rate (around 60 fps) in all the sections of the gameplay, even on older hardware[2].
- For Developers shall be possible to easily develop and swap the minigames among the ones that best fit the training purposes of the user on top of those already provided.
- It shall be possible to play in full screen mode.
- The window shall be resizable to fit in any kind of screen[3].

1.2 Domain analysis

MTSK-Game must exhibit some *minigames*, the ones supplied by us are:

- *WhacAMole*: where the goal is to crush all the moles that emerge from the dens, before they re-enter them, avoiding to detonate bombs that will also pop up from the burrows.
- *DodgeATriangle*: in which the player has to slide a *rectangle* up and down in a column switching lanes, aiming to avoid moving *triangles*.
- *CatchTheSquare*: where the user should destroy *squares* running over them with a *circle* before their timers runs out. With time, multiple squares will spawn at the same time with an increasing rate.
- *FlappyBirdAlike*: where the user needs to control a *cursor* leading it to avoid *obstacles* that will come towards it.

For the game to end, and the *score* to appear, it'll be sufficient losing in only one of the currently displayed minigames.

[2] e.g. Intel Core i5 (fourth generation), 8Gb of RAM.
[3] Provided a minimum resolution.

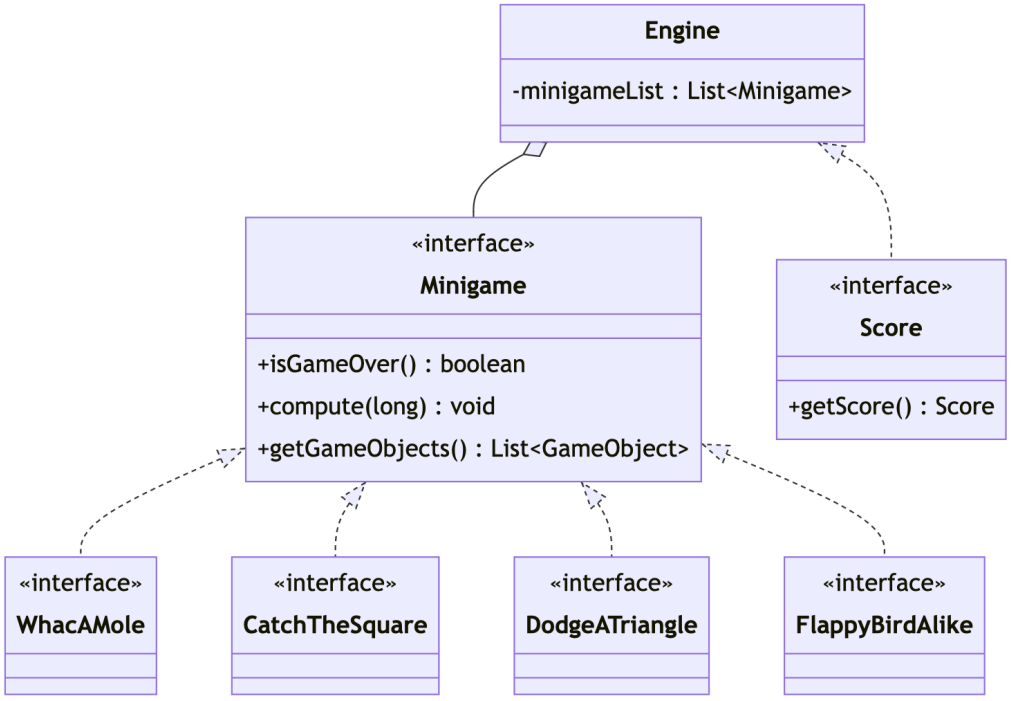

Figure 1.1: UML diagram of the domain analysis

Chapter 2

Design

2.1 Architecture

The project is built on the "model-view-controller" architecture: the **View** will be the entry point of the program. At its creation it will instantiate the game window and the **Controller** class. When the user decides to start the run it will be asked to the controller to create an **Engine**, this class will manage each **Minigame**.
The Model part is represented by the implementation of **Minigame**s these will contain **GameObject**s that will evolve over time with a specific logic. The **Controller** directs the run with the game loop pattern, updating each frame the view with the **GameObject**s fetched from the **Engine**. The **Controller** takes also care of intercepting the user's input from the **View**, then it forwards it to the Engine. Once the input has been received, the Engine is responsible for communicating it to each minigame: these in turn will update the state of all the GameObjects they contain, i.e. every single entity in the various playing fields.

2.2 Detailed design

Problem: Each GameObject must have a specific aspect and behaviour
Solution: Each minigame is composed of **GameObject**s: those items use a component pattern, thanks to which we get a full separation of concerns based on domains (allow a single entity to span multiple domains each other[1]).

[1]From GPP, CH 14

- *PhysicsModel*: Interface that deals with the physical state of a Game Object, moving it according to its speed, considering the environment in which it is located (edges of the field) and the other objects it interacts with (collision with obstacles).
- *AspectModel*: It is the interface that sets the look of the single **GameObject**, specifying which of the Drawing instructions need to be used.
- *InputModel*: Interface related to a single GameObject that reads the input stored in the engine and applies it, if the object recognizes it as its own command, changing its specifications (e.g. coordinates, speed).
- *HitBoxModel*: Defines the shape and sizes of the hitbox that the object shall interpret when colliding with other hitboxes.

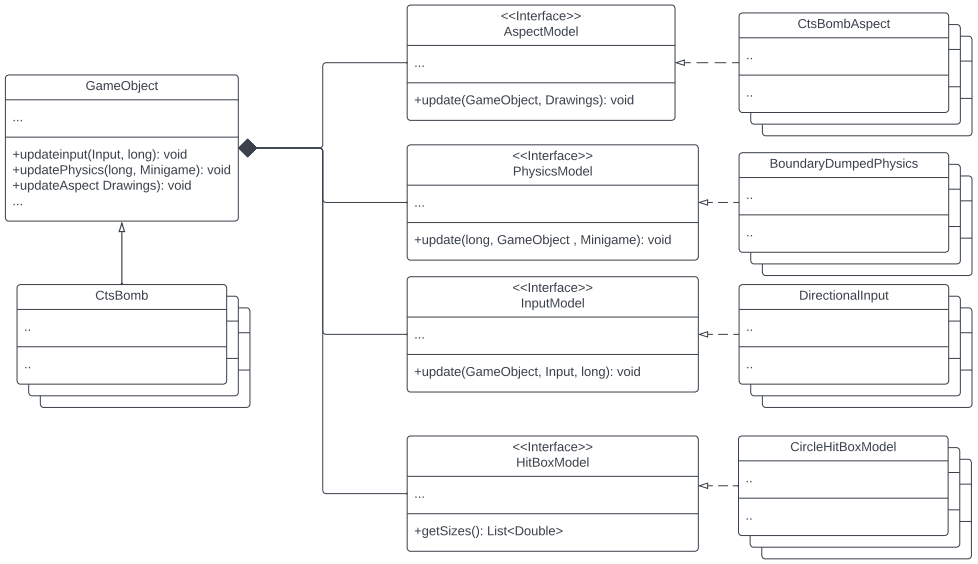

Figure 2.1: UML diagram of the component pattern

Leonardo Tassinari

Difficulty in CTS

Problem: The spawn rate of the squares shall increase over time.

Solution: The CatchTheSquare's constructor adopts the strategy pattern taking a *Function* interface. That function will return to the minigame the number of bombs that are expected to be spawn so far given the total amount of milliseconds elapsed from the beginning of the minigame. My implementation of the function uses an exponential curve to increment the spawn rate progressively until a certain rate. The limit rate is represented by the inclination of the line tangent to the exponential curve on the point where the derivate of the exponential reaches that steepness.

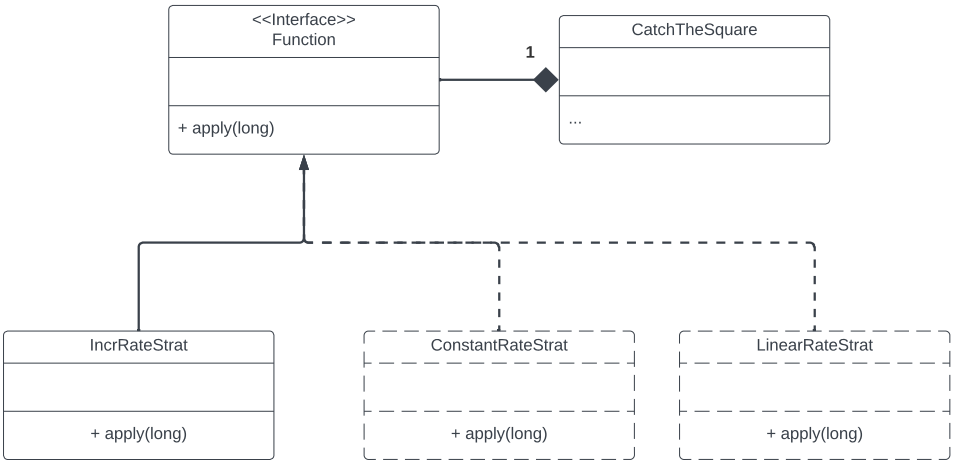

Figure 2.2: The corresponding UML diagram

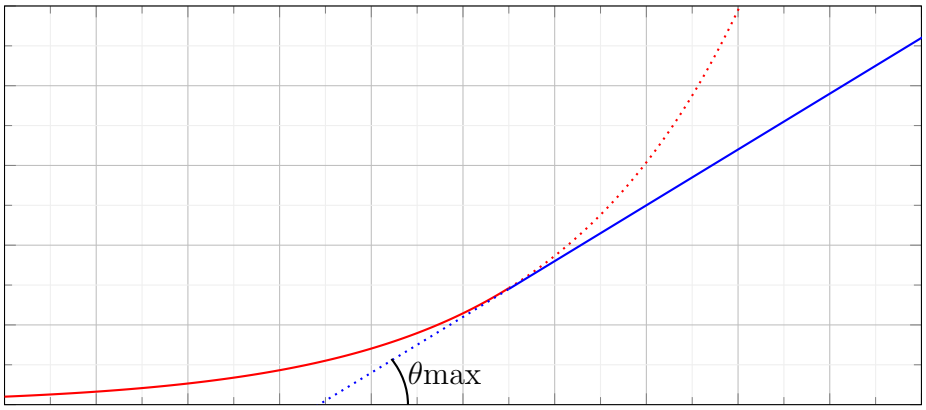

Figure 2.3: An example of the trend of the implemented function

Problem: The *View* shall display different items over time.
Solution: The *JavaFxView* adopts a state pattern, each state has a method to display its item overwriting the ones present at the moment.
A special state, the *GameState* is a subclass of state able to show the gameplay.

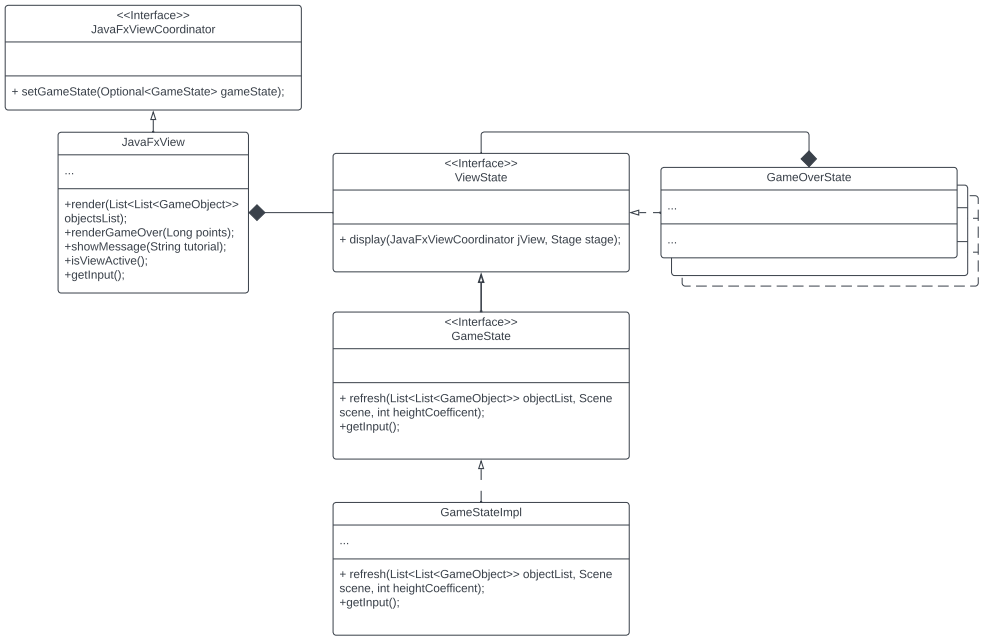

Figure 2.4: The corresponding UML diagram

Pietro Olivi

Difficulty in WhacAMole
Problem: Problem n.1
Solution: When it came time to deal with the Moles and Bombs spawn problem, as well as providing my own version of the algorithm I thought it was vital to come up with a solution that would leave room for future changes, i.e. ideas on how to handle the GameObjects' random extraction. I therefore recognized a natural correspondence with the strategy pattern. Initially I was thinking in terms of time intervals: At the beginning of each set period I would extract a variable number of objects (also assigning the lair from which to spawn) so that they would appear by the end of that interval, however so many checks had to be done that the entire lifecycle of Moles and Bombs did not exceed the right limit of the current interval that the code became unclear. I therefore decided to extract a variable number of objects, at most equal to the maximum number of GameObjects simultaneously in action, and do it again when all of them had finished their work (each entity will also have a waiting time before appearing, otherwise it would become impossible).

Problem: Each Mole and Bomb will have to appear in a specific spot on the pitch, but where?
Solution: The problem of the positioning of the dens is certainly not going to have a single solution, since this strongly depends on the number of dens with which you intend to play. Taking the classic arcade version of Whac-a-Mole as a reference, I provided an implementation that distributes the holes equidistantly (into a table), suitable for numbers whose root is an integer. However, wanting to allow the insertion of other possible positioning strategies, I again leaned on the strategy pattern (although the interface does not appear as a Whac-a-Mole field but as a variable inside the constructor, being used only there).

Lorenzo Dalmonte

Difficulty in FlappyBirdAlike
Problem: Problem n.1
Solution: Solution n.1

Chapter 3

Development

3.1 Automated Tests

Appendix A

User Guide

Catch-The-Square

You will control a circle with the "WASD" keys, moving it on the pane, your objective is to hit the spawing squares to remove them, each square has a timer, don't let it run out!
When you bounce on the walls yours speed will be dumped, therefore you cannot gain high speed and forget this minigame. With time the spawn rate of the squares will increase, good luck!

Flappy-Bird-Alike

The player controls a triangle shaped cursor on the left side of the screen. Press the SPACEBAR to jump and evade obstacles incoming from the other side. You can jump in midair too!

Whac-A-Mole

Moles and Bombs will randomly appear on the playing field: sufficient conditions for losing the game are to let a mole return to its burrow without being crushed, or to hit a bomb. If you are using a keyboard with horizontal arrangement of numbers, to hit an object that is outside the hole, just press the number corresponding to the latter. In particular, the dens will be numbered in ascending order starting from the first in the upper left, continuing towards the right, and then moving on to the first in the next row (once the previous one is finished). If you have a numpad available then, in the case of the standard 9-hole version of the game, you will have to press, more intuitively, the key in the position corresponding to that of the lair.

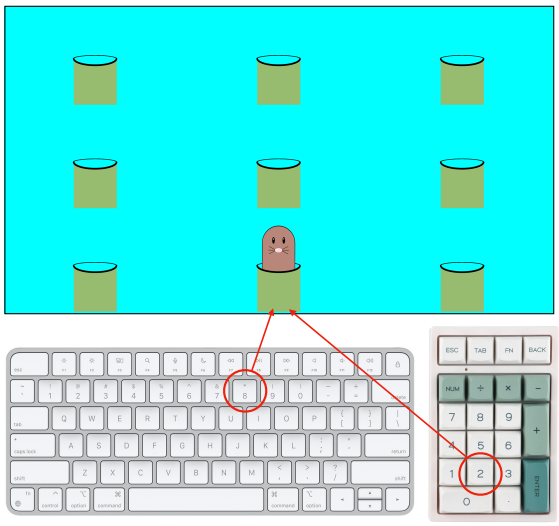

Figure A.1: Keys to use to play Whac-a-mole

Dodge-A-Triangle

The player controls a square that moves up and down lanes by pressing the UP and DOWN arrows respectively on the keyboard.
Avoid triangle enemies coming from the sides of the screen.

Appendix B

Developer Guide

Creation of a new minigame

To create a new *Minigame* you can implement its interface. Each minigame will keep an internal list of *GameObjects*, that will be updated each frame in 3 main phases:

- input process
- update
- render

Each GameObject is composed of Modules regarding different aspects of the object. You can create your own modules or use the ones already present, take a look at the javaDoc for a more comprehensive list.

Input Process

Update

Render

Appendix C

Lab Practices

pietro.olivi2@studio.unibo.it

- Lab 05: https://virtuale.unibo.it/mod/forum/discuss.php?d=114647#p169720
- Lab 06: https://virtuale.unibo.it/mod/forum/discuss.php?d=115548#p171327
- Lab 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=117044#p173090
- Lab 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=118995#p175807

lorenzo.dalmonte4@studio.unibo.it

- Lab 06: https://virtuale.unibo.it/mod/forum/discuss.php?d=115548#p170779
- Lab 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=117044#p172448
- Lab 08: https://virtuale.unibo.it/mod/forum/discuss.php?d=117852#p173898
- Lab 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=118995#p175820
- Lab 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=119938#p176290

leonardo.tassinari6@studio.unibo.it

- Lab 04: https://virtuale.unibo.it/mod/forum/discuss.php?d=113869#p169260
- Lab 05: https://virtuale.unibo.it/mod/forum/discuss.php?d=114647#p169543
- Lab 06: https://virtuale.unibo.it/mod/forum/discuss.php?d=115548#p171093
- Lab 07: https://virtuale.unibo.it/mod/forum/discuss.php?d=117044#p172618
- Lab 08: https://virtuale.unibo.it/mod/forum/discuss.php?d=117852#p173682
- Lab 09: https://virtuale.unibo.it/mod/forum/discuss.php?d=118995#p174740
- Lab 10: https://virtuale.unibo.it/mod/forum/discuss.php?d=119938#p176083
- Lab 11: https://virtuale.unibo.it/mod/forum/discuss.php?d=121130#p177362
- Lab 12: https://virtuale.unibo.it/mod/forum/discuss.php?d=121885#p178213