Report
"MTSK-GAME"

Pietro Olivi
Leonardo Tassinari
Lorenzo Dalmonte
Alessio Paoloni

February 10, 2023

Contents

Chapter 1

Analysis

The software commissioned by the professors of the Object Oriented Programming course aims to create an application designed to enhance psychomotor 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.
  A minigame is a potentially-never-ending challenge that requires simple actions from the player in order to keep the game going. After a short amount of time a new minigame will appear and so on until all four minigames are shown.
- The player's goal is to survive as long as possible. After failing a minigame the application will display the final result, therefore the software has to keep track of how long the player has lasted in the match in order to calculate the score.

Non functional

- The application shall sustain high framerate (around 120 fps) in all the sections of the gameplay, even on older hardware[1].

[1] e.g. Intel Core i3 (fourth generation), 4Gb of RAM.

- It 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.

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 dug holes, before they re-enter them.
- *DodgeATriangle* in which the player has to slide a *rectangle* up and down switching lanes, aiming to avoid hitting moving *triangles*.
- *CatchTheSquare* where the user should collect *squares* running over them with a *circle* before they disappear.
- *FlappyBirdAlike* where the user needs to control a *cursor* leading it to fit between *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 mingames.

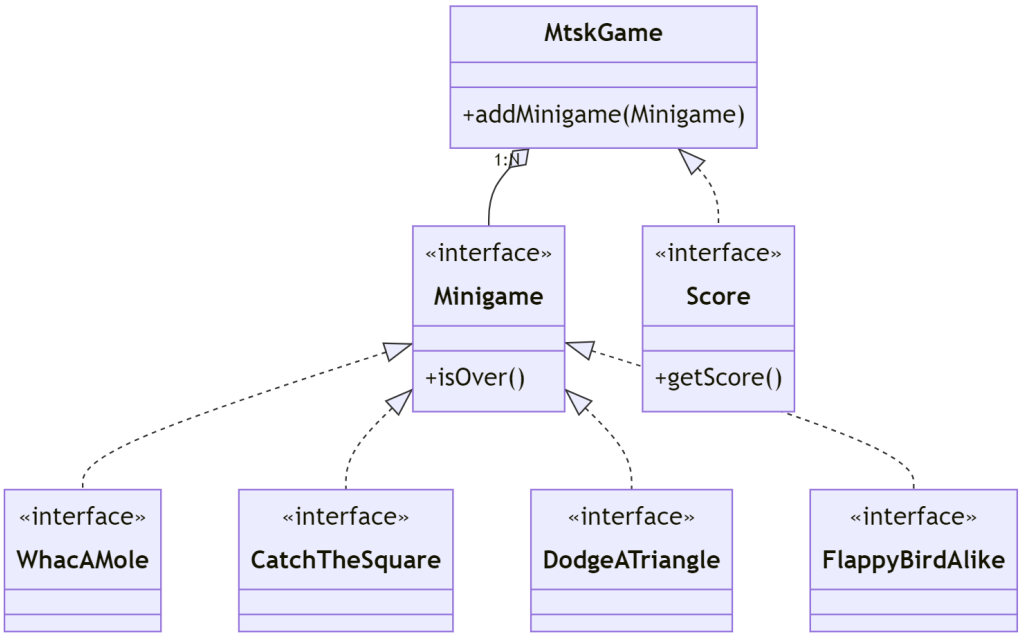

Figure 1.1: UML diagram of the domain analysis

Chapter 2

Design

2.1 Architure

We decided to use the "model-view-controller" architecture: the **Engine** class rapresents the controller and the entry point of the program. From there we create an instance of the view, **SwingView** in our case, and the different minigames where each contains the relative model.

2.2 Detailed design

Each minigame is composed of **GameObject**s, those items use a component pattern:

- PhysicsModel: the model that dictates how the object fields evolve over time.
- AspectModel: specifise aspect parameter of the object.
- InputModel: updates the fields according the inputs.