Bìa

* Thành viên nhóm
* Tên trường, tên project, tên lớp

Giới thiệu

Lời cảm ơn

Table of Contents

[**1.** **Introduction to Game RushRelic (mordern remake of RushHour)** 5](#_Toc202979005)

[**a.** **Installation** 5](#_Toc202979006)

[**i.** **Github** 5](#_Toc202979007)

[**ii.** **Zipped file** 5](#_Toc202979008)

[**iii.** **Video Youtube** 5](#_Toc202979009)

[**b.** **Interface** 5](#_Toc202979010)

[**i.** **Screens layout and flow** 5](#_Toc202979011)

[**ii.** **Gameplays** 5](#_Toc202979012)

[**c.** **Design system and Design pattern** 5](#_Toc202979013)

[**i.** **System Class** 5](#_Toc202979014)

[**ii.** **Core Game Classes: Map and Vehicle** 7](#_Toc202979015)

[**2.** **Description of the Maps** 8](#_Toc202979016)

[**a.** **Foudational Elements in a Map** 8](#_Toc202979017)

[**b.** **Map Details** 8](#_Toc202979018)

[**i.** **Map 1** 8](#_Toc202979019)

[**ii.** **Map 2** 8](#_Toc202979020)

[**iii.** **Map 3** 8](#_Toc202979021)

[**iv.** **Map 4** 8](#_Toc202979022)

[**v.** **Map 5** 8](#_Toc202979023)

[**vi.** **Map 6** 8](#_Toc202979024)

[**vii.** **Map 7** 8](#_Toc202979025)

[**viii.** **Map 8** 8](#_Toc202979026)

[**ix.** **Map 9** 8](#_Toc202979027)

[**x.** **Map 10** 8](#_Toc202979028)

[**3.** **Search Algorithm Description** 8](#_Toc202979029)

[**a.** **Input and Output Specification** 8](#_Toc202979030)

[**b.** **State Representation and Visited Table** 8](#_Toc202979031)

[**c.** **Solvers explaination** 8](#_Toc202979032)

[**i.** **BFS Solver** 8](#_Toc202979033)

[**ii.** **DFS Solver** 8](#_Toc202979034)

[**iii.** **UCS Solver** 8](#_Toc202979035)

[**iv.** **A\* Solver** 8](#_Toc202979036)

[**4.** **Experiments** 8](#_Toc202979037)

[**a.** **Search Time** 8](#_Toc202979038)

[**i.** **Measurements** 9](#_Toc202979039)

[**ii.** **Visualizations** 9](#_Toc202979040)

[**iii.** **Insight** 9](#_Toc202979041)

[**b.** **Memory Usage** 9](#_Toc202979042)

[**i.** **Measurements** 9](#_Toc202979043)

[**ii.** **Visualizations** 9](#_Toc202979044)

[**iii.** **Insight** 9](#_Toc202979045)

[**c.** **Expanoded** 9](#_Toc202979046)

[**i.** **Measurements** 9](#_Toc202979047)

[**ii.** **Visualizations** 9](#_Toc202979048)

[**iii.** **Insight** 9](#_Toc202979049)

[**5.** **Project Plan and Task Distribution** 9](#_Toc202979050)

[**a.** **Project Plan** 9](#_Toc202979051)

[**b.** **Task Distribution** 9](#_Toc202979052)

[**6.** **References** 9](#_Toc202979053)

[**7.** **Apendix** 9](#_Toc202979054)

1. **Introduction to Game RushRelic (mordern remake of RushHour)** 
   1. **Installation** 
      1. **Github**
      2. **Zipped file**
      3. **Video Youtube**
   2. **Interface**
      1. **Screens layout and flow**
      2. **Gameplays**
   3. **Design system and Design pattern**
      1. **System Class** 
         1. Class Program

The game is built using **pygame**, a Python library for 2D game development that supports graphics rendering, user input, and sound.

The main function is the core of the system, structured as a continuous game loop. Each loop **clears the screen**, **processes input**, **updates object states**, and **redraws the frame**.

To ensure only one controller instance, the **Singleton** pattern is applied to main.

Rendering uses **double buffering**: objects are first drawn to a **back buffer**, then flipped to the screen for smooth display and reduced flickering.

* + - 1. Class Screen

This class applies the **State design pattern** to manage different UI screens. New screens can be added using the **addScreen(‘hello\_screen’)** method, and the current screen can be switched using **setScreen(‘hello\_screen’)**.

* + - 1. Class Graphic

This class also follows the **Singleton** design pattern. It centralizes all rendering-related functions using pygame, ensuring that the library is imported and used only once across the entire source code.

* + - 1. Class Solver

The Solver class is a core component responsible for implementing the game's solving algorithms. It includes a **BaseSolver** class that contains shared logic and helper functions used across all strategies, ensuring consistency and reducing code duplication.

To support scalability, a **SolverFactory** is used to create solver instances dynamically, allowing new algorithms to be added with minimal changes.

The system adopts the **Strategy design pattern**, enabling the program to switch between different solving algorithms (e.g., BFS, DFS) at runtime. Each strategy is encapsulated in a separate class, making the design modular, extensible, and easy to maintain.

* + - 1. Class relative to UI

This module contains classes such as **Button**, **Text**, **Font**, and **Mouse**, which are responsible for rendering the game’s user interface.

The game exclusively ues the **Vinizinho** font for all text rendering, ensuring a consistent visual style.

The Button class applies the **State** design pattern to manage its visual states, including default, hover, and click. This approach improves the clarity and maintainability of UI behavior.

* + - 1. Class AudioManager

AudioManager is a Singleton class responsible for handling all audio in the game, including background music and sound effects (e.g., mouse clicks).

It is designed with multithreading support to ensure that audio playback does not interrupt other game actions or reduce performance.

To enhance the user experience, the game includes a settings screen where players can toggle music on/off and adjust volume levels.

* + - 1. Class ResourceManager

ResourceManager is a class responsible for initializing and managing game assets, especially images.

There are two types of images handled: static images and sprites.

* Static images are used for non-animated objects such as maps and backgrounds.
* Sprite images are used for animated elements like the Character class.

This centralized asset management improves performance, memory usage, and makes resource loading more consistent across the game.

* + 1. **Core Game Classes: Map and Vehicle**

This section defines two essential classes — **Vehicle** and **Map** — which directly affect the game logic and solving process.

* + - 1. Class Vehicle

The Vehicle class represents a car on the map and is defined by five key input parameters:

* **orient**: the orientation of the vehicle (horizontal or vertical)
* **len**: the length of the vehicle
* **x, y**: the vehicle’s coordinates on the grid
* **name**: the name/identifier of the vehicle

A notable method of this class is:

* **copy**(): creates a deep copy of the vehicle, which is essential for generating new states during the solving process.
  + - 1. Map

The Map class manages the game state and solving logic for each level. Key input attributes include:

* **initial\_vehicles**: an array of Vehicle instances loaded from a map.txt file
* **current\_level**: the current level ID used to load the corresponding map file
* **solver**: the selected algorithm for solving the level (through Strategy Solver)
* **solving**: a boolean flag indicating whether the level is currently being solved
* **solution\_moves**: total number of moves in the computed solution
* **current\_move\_index**: used to manage pause/resume functionality
* **move\_timer**: tracks how long the solver takes to finish
* **move\_delay**: controls the animation speed for each move
* **list\_solver**: stores the names of visited vehicles (used for visualizing or debugging)

Important methods include:

* **load\_level\_data\_from\_file**(): reads level data from the file based on current\_level
* **reset**(): resets the map to its initial state
* **update**(): progresses the solving state frame by frame
* **start\_solving**(): selects and runs the appropriate algorithm (e.g., if BFS is selected, it will use BFSStrategy to solve the puzzle; the map instance itself is passed as the input)
* **print\_solution**(): displays the list of moves that solve the level
* **handle\_pause**(): toggles between pause and resume using the solving flag

1. **Description of the Maps** 
   1. **Foudational Elements in a Map**

Each map in *RushRelic* is designed based on a consistent set of foundational elements that define the gameplay experience. These include:

* 1. **Map Details**
     1. **Map 1**
     2. **Map 2**
     3. **Map 3**
     4. **Map 4**
     5. **Map 5**
     6. **Map 6**
     7. **Map 7**
     8. **Map 8**
     9. **Map 9**
     10. **Map 10**

1. **Search Algorithm Description** 
   1. **Input and Output Specification**
   2. **State Representation and Visited Table**
   3. **Solvers explaination**
      1. **BFS Solver**
      2. **DFS Solver**
      3. **UCS Solver**
      4. **A\* Solver**
2. **Experiments** 
   1. **Search Time**
      1. **Measurements**
      2. **Visualizations**
      3. **Insight**
   2. **Memory Usage**
      1. **Measurements**
      2. **Visualizations**
      3. **Insight**
   3. **Expanoded** 
      1. **Measurements**
      2. **Visualizations**
      3. **Insight**
3. **Project Plan and Task Distribution** 
   1. **Project Plan**
   2. **Task Distribution**
4. **References**
5. **Apendix**