CMPE 492
SENIOR PROJECT 2

Project Name
Astroni

Project Supervisor
Ulaş GÜLEÇ

Project Team Members
Ata AYYILDIZ
Nilay ÖZKIR
Ömercan YALÇIN

Project Jury Members
Emin KUĞU
Gökçe Nur YILMAZ

Project Website
https://teduastroni.github.io

Table of Content

1 Introduction

1.1 Object design trade-offs

1.1.1 User-Friendliness vs Game difficulty

Since Astroni is an action game that aims to have fun, it does not contain any elements that will challenge the user. Additionally, since excessive difficulty in the game will distract the player from the game, we will design the difficulty level to make the user feel as comfortable as possible. Additionally, in an action game, there will be no difficulty that will drive the player away, as excessive difficulty levels will drive the player away.

1.1.2 Visual and Audio Richness vs Performance

Astroni will have various visual and audio effects. With this diversity, we aim for players to enjoy the game more. The objects or auditory elements we will use in the game will further draw the player's sensory organs into the game and increase the pleasure the players will get from the game. We also aim to make the game world look deeper thanks to the lighting and shading effects we will use in the game. However, these rich contents can negatively affect performance in some cases. That's why we make sure that the effects we use do not have a negative impact on game performance. For example, we take care not to use visual or audio effects that will seriously reduce the visuality.

1.1.3 Guidance and Tips vs Free Exploration

Since Astroni aims to help its users discover the secret world of Ador, there will be many directions and tips about the game. The game will feature free exploration, but the player will be able to progress through the game with more clues. For example, the player will be able to find our lost star Astroni more easily with clues. In addition, thanks to the tips, the player will enjoy the game more. Because he will make some kind of observation while trying to find these clues.

1.1.4 Easy Accessibility vs Game Mechanics Complexity

Astroni will be as mechanically uncomplicated as possible to make the game quickly understandable and playable by any player. Because the complexity of the game mechanics

makes it difficult to have fun and can distract the players from the game. In addition, the mechanics we will use in the game will be designed at a level that every player can easily use. In this way, we aim to increase the playability of the game by everyone. For example, the mechanics we will use in the game will be simple mechanics such as moving and shooting arrows.

1.2Interface documentation guidelines

For internal packages:

Class Name, implements I list
vis attribute: type
vis operation (arg list): return type

Where:

- vis = visibility (‘+’ for public, ‘#’ for protected, ‘-‘ for private, ‘,’ for separating get, set)
- attribute = data member aka field aka instance
- operation = method

We use this template for our interfaces, other than that Underlines means that attribute or operation is static. We use Pascal naming method for all our code. We explain functionalities of everything under the table title by title.

For external packages:

Description

URL: docs.unity3d.com/…

URL is the url to the documentation of that specific class.

We use this template for external packages since there are too complex to add by all their attributes and operations.

1.3 Engineering Standards

To design a 2D pixel art adventure game for PC, developers adhere to a set of standards that ensure quality and efficiency throughout the development process. This involves

choosing an appropriate game engine, such as Unity, and creating a modular code architecture to manage game systems effectively. Optimization techniques such as object pooling and texture atlasing are used to improve performance, while a fluid art pipeline seamlessly integrates pixel art assets into the game. Designing intuitive user interfaces, responsive input processing and impressive audio integration are crucial for a captivating player experience. Extensive testing, debugging and documentation ensure a stable and polished final product that is compatible with various platforms and accessible to different audiences. By following these engineering standards, developers can create engaging and memorable 2D pixel art adventures for PC gamers.

1.3.1 PC Development Standards

PC development standards encompass a comprehensive set of practices aimed at ensuring the efficient and high-quality development of software for personal computers. These standards cover various aspects of the development lifecycle, including coding conventions, version control utilization, software architecture design, performance optimization techniques, user interface principles, testing methodologies, security measures, localization efforts, and documentation practices. By adhering to these standards, developers strive to create desktop games that are reliable, scalable, user-friendly, secure, and accessible to global audiences. From maintaining clean and consistent codebases to implementing robust testing strategies and providing thorough documentation and support, following PC development standards is essential for delivering software that meets the needs and expectations of PC users.

1.3.2 Standards for Game Design and Development

Developing a 2D top-down pixel art game in Unity requires adherence to specific standards to ensure a smooth and efficient development process. Initially, it's crucial to establish a clear game concept, outlining the game's mechanics, setting, and visual style.

Prototyping key gameplay elements early on allows for iteration and refinement based on playtesting feedback. Art and audio direction play a significant role, with pixel art assets and chiptune-style music contributing to the game's nostalgic charm. Technical implementation involves using Unity's 2D tools to create tilemaps, sprite animations, and efficient collision detection systems. Adhering to coding best practices and optimization techniques ensures optimal performance, particularly on lower-end devices. User experience design focuses on intuitive controls and clear visual feedback, enhancing player immersion. Narrative elements, if included, should be seamlessly integrated into gameplay mechanics. Thorough playtesting and bug testing are essential to identify and address issues before release. Finally, deploying the game to various platforms and providing post-launch support and updates complete the development process, resulting in a polished and enjoyable pixel art experience for players.

1.3.3 Standards Particular to PC

IEEE has initiated standard projects to address issues of interoperability and compatibility in the global gaming services industry, focusing on mobile gaming performance evaluation and optimization. IEEE P2861 and IEEE P2861.1 involve stakeholders such as game developers, publishers, network vendors, and device vendors. IEEE P2861, titled "Standard for Mobile Gaming Performance Evaluation and Optimization," aims to enhance gamers' experience by balancing high demands from game applications with resource limitations on mobile devices. It seeks to establish criteria and mechanisms for optimizing game performance and device efficiency, along with evaluating game fluency and optimization effects. This standard project also outlines gaming scenario classifications and defines interfaces between game applications and devices, ensuring a smoother and more standardized gaming experience across various platforms.

1.4 Glossary; definitions, acronyms and abbreviations

PC: Personal Computer

Player : User of video game software

vis : Visibility (‘+’ for public, ‘#’ for protected, ‘-‘ for private, ‘,’ for separating get, set)

Attribute : Data member aka field aka instance

Operation : Method on script

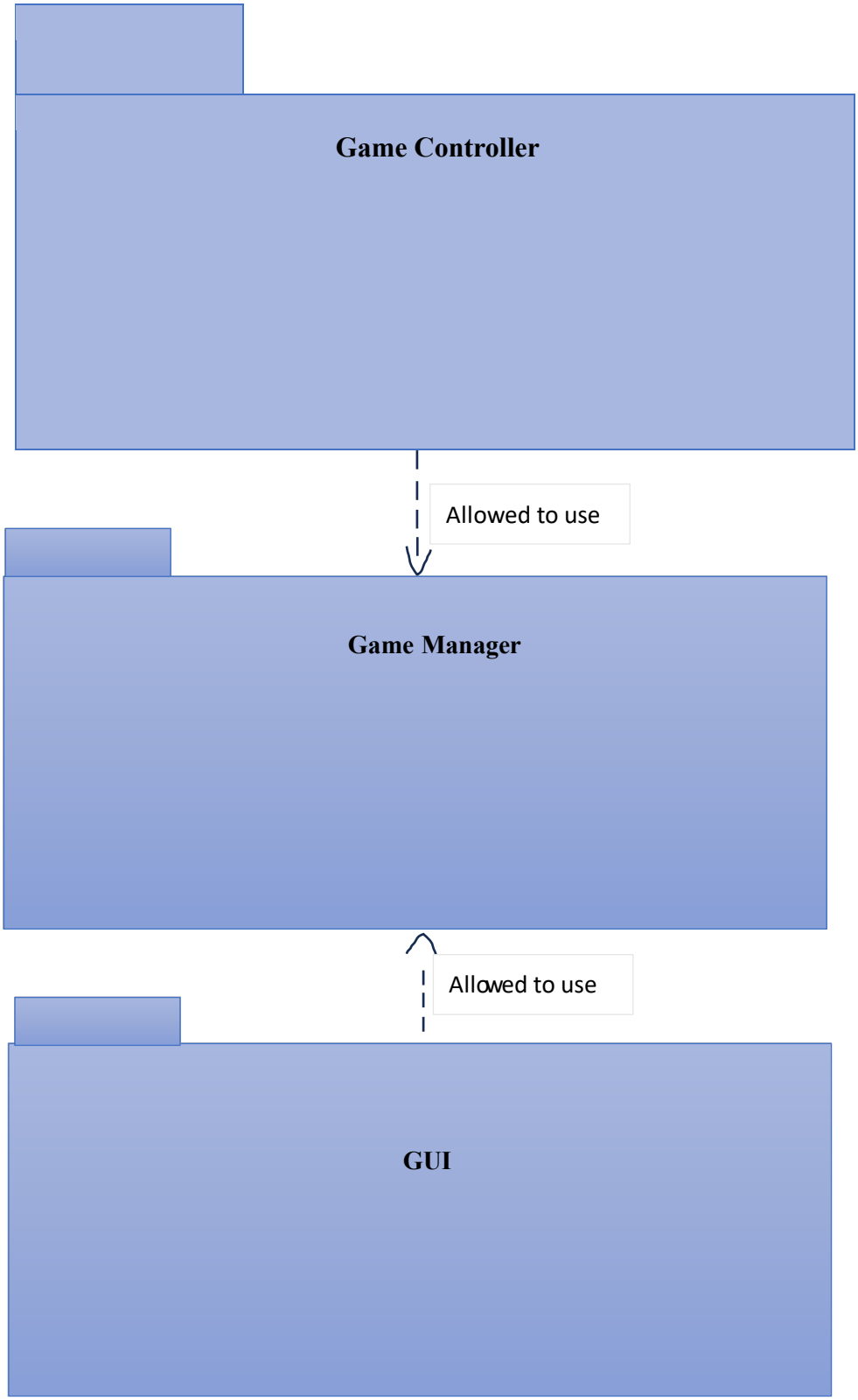
URL : Uniform Resource Locator. A URL is nothing more than the address of a given unique resource on the Web.

IEEE : Institute of Electrical and Electronics Engineers - Professional organizations company

GUI : Graphical User Interface, layer that enables a person to communicate with a computer through the use of visuals.

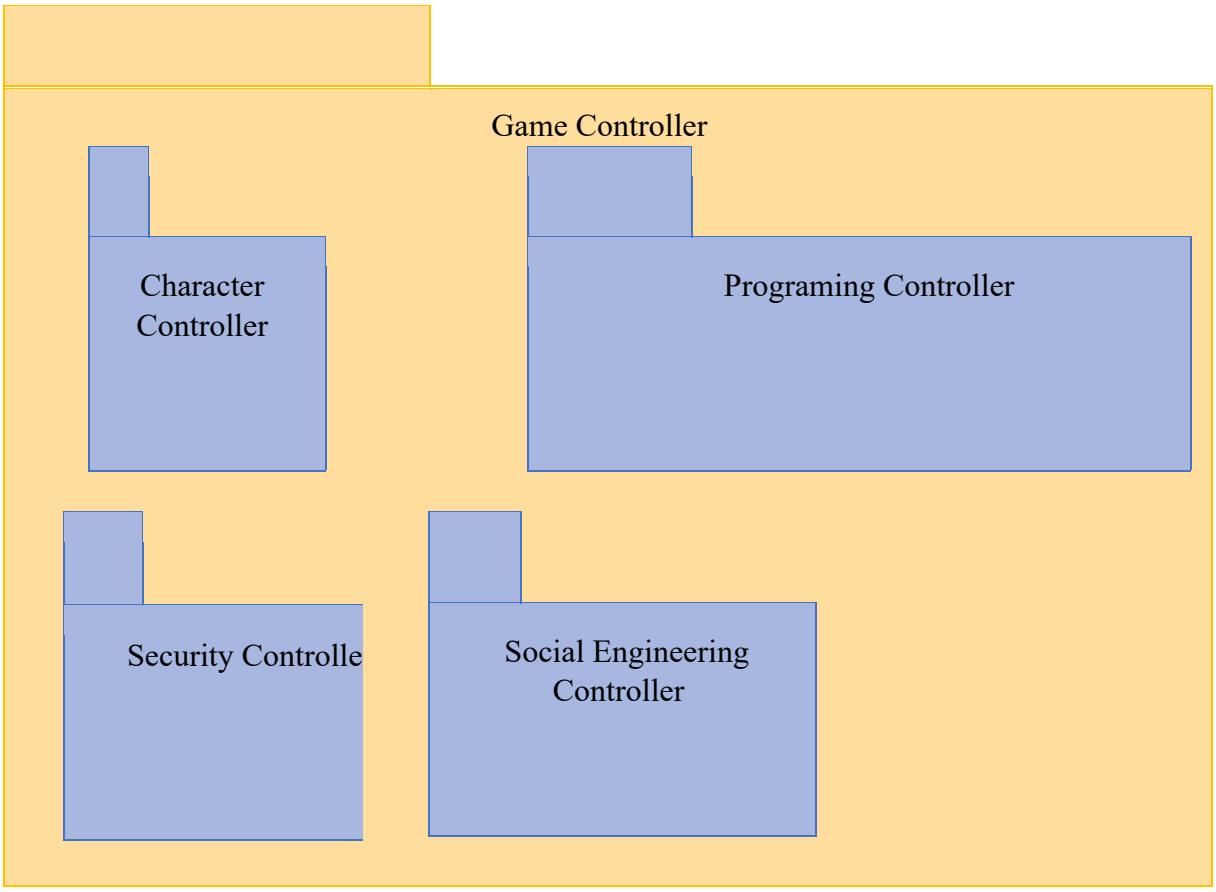
2. Packages

This section presents the packages of our project. Astroni has three packages which are GUI, Game Controller and lastly Game Manager.



GUI and Game Controller are the top levels of our packages and these provides interaction between user and game. Game Controller's main goal is taking input from user and allow to control mechanics, essentials and progress of game according to input, preferences or progress. GUI's main goal is showing the desires of user like options, interactable objects and much more. Also GUI is a way to interact some mechanics with visuals.

2.1 Game Controller



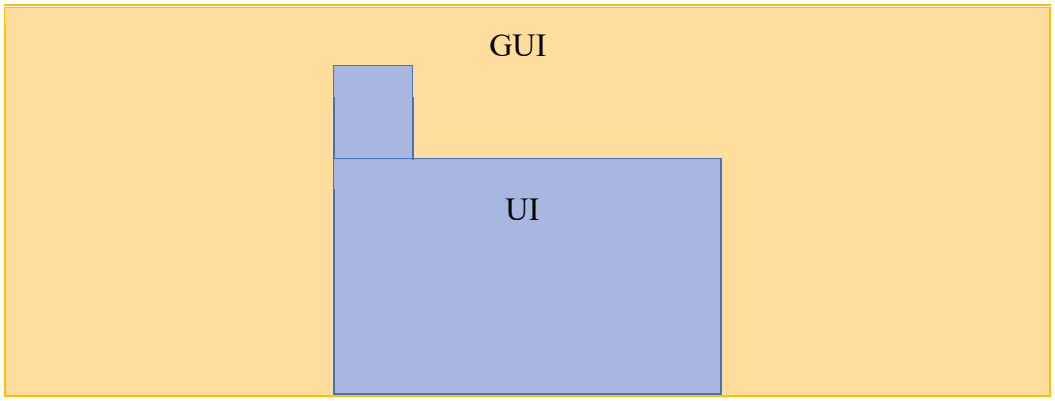
Character Controller is for control the behavior of character.

Security Controller is for the control NPCs in game.

Programing Controller is for the control programing mechanics in game.

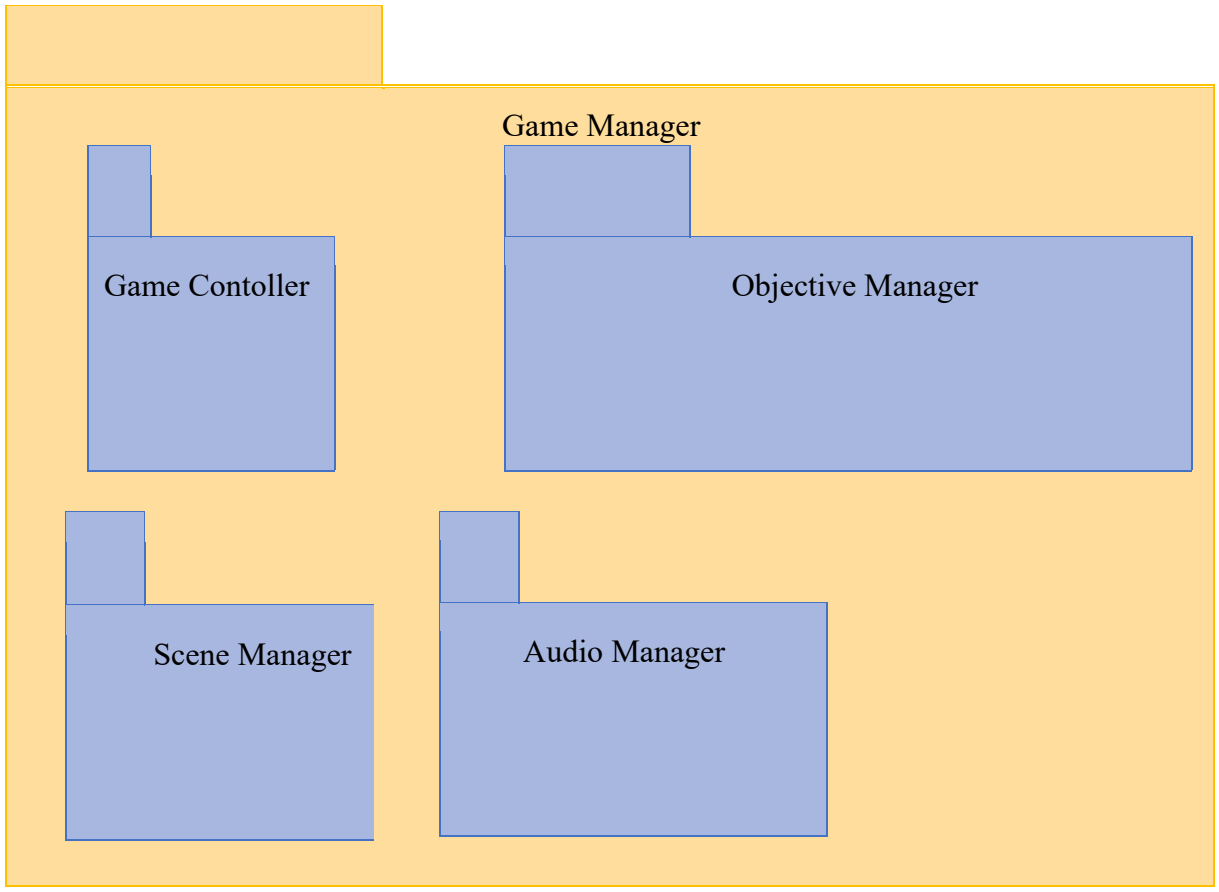
Social Engineering Controller is for the control objectives and progress in social engineering gameplay.

2.2 GUI



UI is for the control user interface, visuals and their access to the game manager.

2.3 Game Manager



Game Controller is the class for control progress in game.

Objective manager is for the control requirements of objectives and progress in missions.

Scene Manager is controlling class for scene which is currently playing.

Audio Manager is for control audios in game such as sound effects and music.

3 Class Interfaces

3.1 Scenario Control System

ICharacter
GetIsDead(): bool Move(): void GetFightable(): bool GetName(): string Kill(): void TakeDamage(): void

GetIsDead(): This method is used to check if the character is dead or alive. It returns a boolean value indicating whether the character is dead (true) or alive (false).

Move(): This method is responsible for moving the character. The specifics of how the character moves would depend on the implementation, such as walking, running, or any other form of locomotion.

GetFightable(): This method determines whether the character can engage in combat. It returns a boolean value indicating whether the character is capable of fighting (true) or not (false).

GetName(): This method retrieves the name of the character as a string. This can be useful for displaying the character's name in user interfaces or for other identification purposes.

Kill(): This method is used to kill the character. It likely sets the character's state to dead and performs any necessary cleanup or actions associated with the character's death.

TakeDamage(): This method is called when the character takes damage. It handles reducing the character's health or applying any other effects associated with taking damage.

3.2 PC Controller

It is a vital interface designed specifically for facilitating player interaction within two-dimensional game environments on personal computer (PC) platforms. This interface orchestrates a comprehensive array of functions tailored to the unique requirements of navigating and interacting within 2D game worlds. Through intuitive methods such as movePlayer, players can seamlessly direct the protagonist's movements across the screen, employing directional inputs to traverse the game's terrain with precision and fluidity.

Additionally, the controller's interact function enables players to engage with various in-game elements, including objects, NPCs, and environmental triggers, fostering immersive gameplay experiences. Efficient inventory management is facilitated by the manageInventory method, allowing players to access, manipulate, and utilize items stored within their inventory with ease.

Furthermore, the controller's pauseGame feature empowers players to temporarily suspend gameplay, accessing menus and options seamlessly integrated into the PC interface. To ensure player progress is safeguarded, the saveGame functionality permits players to record their current state at any juncture, preserving achievements and advancements for subsequent play sessions.

3.3 Input Action Manager

It serves as a pivotal component in managing and coordinating various player inputs within a game environment. This manager is responsible for interpreting and processing input signals from different input devices, such as keyboards, mice, gamepads, or touchscreens, and translating them into actionable commands within the game. Through a systematic approach, the Input Action Manager maps input signals to specific in-game actions or functions, allowing for seamless and intuitive player interaction. This mapping process involves configuring input bindings, defining input actions, and handling input events with precision and efficiency.

Additionally, the Input Action Manager may incorporate features such as input buffering, input prioritization, and input remapping to ensure optimal responsiveness and flexibility in accommodating diverse player preferences and input configurations. By providing a centralized and adaptable framework for managing player inputs, the Input Action Manager plays a critical role in enhancing the overall gameplay experience, promoting accessibility, and facilitating immersive engagement across a wide range of gaming platforms and devices.

4 References

https://docs.unity3d.com/Manual/Unity2D.html

https://standards.ieee.org/develop/

https://ieeexplore.ieee.org/document/9269032