Cuadrado

Dumb Dino Project
Software Architecture Document

Version <1.0>

Revision History

Date	Version	Description	Author
<22/Oct/22>	<1.0>	First Version	Nikhil Singla, Robert Froeschl

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2 Scope

This Software Architecture Document provides an architectural overview of the Cuadrado Game. Cuadrado allows puzzle-loving players to enjoy within a defined software game on their computers They can also compete with each other, or the AI to see who is the fastest player.

1.3 Definitions, Acronyms, and Abbreviations

See Glossary, document upedu_ex_gloss.pdf

1.4 References

1. Glossary
2. Use case specification
3. Supplementary Specification

1.5 Overview

This document will explain the overall decomposition of the design modal, the packages used in the game, and the classes contained within the code.

2. Architectural Representation

This document presents the architectural as a series of views; use case view, process view, deployment view, and implementation view. These views are presented as Rational Rose Models and use the Unified Modeling Language (UML).

3. Architectural Goals and Constraints

The Cuadrado system to be developed is a maze game that is downloaded onto one's personal device. It consists of four major components: a server module, a player package, an AI package, and a game package.

All components must execute and be downloaded on a personal device.

The Server and the package components should be located on the same host.

4. Use-Case View

The Use Case View is important input to the selection of the set of scenarios and/or use cases that are the focus of an iteration. It describes the set of scenarios and/or use cases that represent some significant, central functionality. It also describes the set of scenarios and/or use cases that have a substantial architectural coverage (that exercise many architectural elements) or that stress or illustrate a specific, delicate point of the architecture.

4.1 Use-Case Realizations

Refer Use Case Realization document - upedu_ex_uiprt.pd

5. Logical View

This section describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages. And for each significant package, its decomposition into classes and class utilities.

5.1 Overview

This subsection describes visually the overall decomposition of the design model in terms of its package hierarchy and layers.

5.2 Architecturally Significant Design Packages

Design Model: Packages Diagrams

The design model represents explicitly the structure and organization of the Cuadrado game system. Packages and corresponding classes are presented with a brief description.

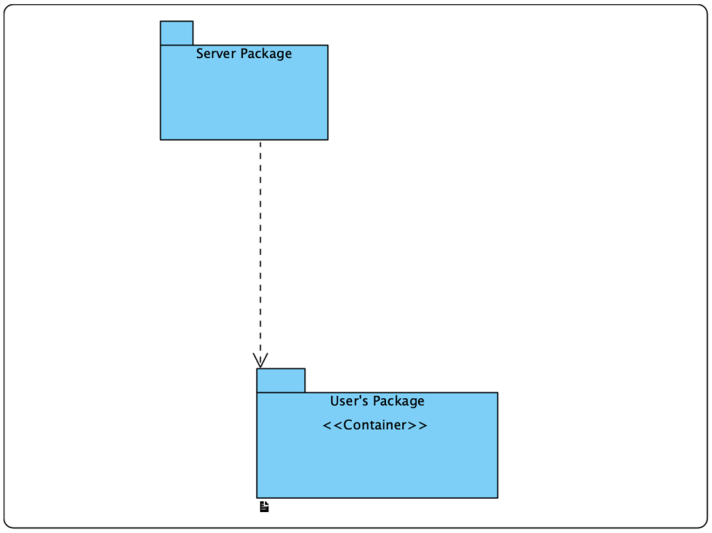

Figure 1: Design Model Packages Level 1

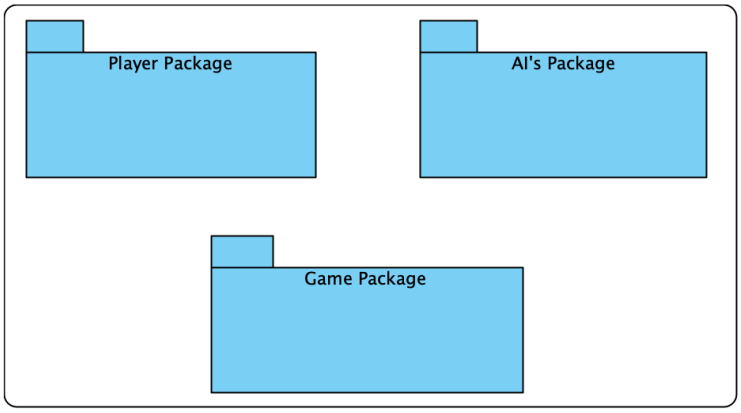

Figure 2: Design Model Packages Level 2 (Users Packages)

Level 1 Packages:

Server Package	
Description:	Main System Package. Although this package is dependant on other system packages, this package is the central point for accessing the game. All client requests are handled by this package
Corresponding classes:	Observer AdministratorObserver
Relations:	Main Cuadrado package. Dependant of: Player, AI, and Game packages
Sub packages:	Users

User Package	
Description:	Container Package for the AI, Player and Game Modules
Corresponding classes:	Is a sub package of the main package Server
Sub packages:	None

Level 2 Packages:

AI	
Description:	This package corresponds to the AI Module. All information and methods regarding AI functions are contained within this package.
Corresponding classes:	Character aiCharacter
Relations:	Is a sub-package of Users.
Sub packages:	None

Player	
Description:	This package corresponds to the Player Module. All information and methods regarding player actions are contained within this package.
Corresponding classes:	Character playerCharacter
Relations:	Is a sub-package of Users.
Sub packages:	None

Game	
Description:	This package corresponds to the Game Module. All information and methods regarding Game functions are contained within this package.
Corresponding classes:	Variables Board Buttons
Relations:	Is a sub-package of Users.
Sub packages:	None

Property	Description
Name	Observer
Description	Implementation of the actions taken by user
Responsibilities	Lets the user interacts with the server to download and update executable
Methods	**ExecuteReq()** : Execute the request **ReceiveReq()** : Listens to coming request **ValidateCAPTCHA()** : Validates the captcha
Attributes	**DownloadLink: Stores the Download Link**
Special Requirements	None

Property	Description
Name	AdministratorObserver
Description	Implementation of the actions taken by server admins
Responsibilities	Lets the server admins interacts with the backend of the server to update executable and manage serverload
Methods	**Update()** : Update the download link **CheckStatus()** : Update status of server **CAPTCHA()** : Set the captcha test method **stopALL("password")** : Killswitch for emergencies with a special code parameter
Attributes	**Credentials: Stores the Admin Login credentials** **status: Stores current server status** **password: Store killswitch password**
Special Requirements	None

Property	Description
Name	Character
Description	Abstract class for how the characters interacting with the game class would behave
Responsibilities	Define actions for all characters Prevent illegal moves Keep track of character stats
Methods	**Move()** : Lets the character move in one of the 4 possible directions **CheckMove()** : Check if move is allowed or not **StatUpdate()** : Updates the stats of the character
Attributes	**start: Stores the Game start value** **end: Stores Game end value** **stats: Custom structure storing all character stats**
Special Requirements	None

Property	Description
Name	aiCharacter
Description	AI Character implementation for character class
Responsibilities	Chose AI move according to algorithm Keep track of AI difficulty Perform the move operation
Methods	**setMove()** : Performs move according to algorithm input **nextMove()** : Returns next move **algorithm()** : Selects and Runs algorithm based on difficulty to find the solution
Attributes	**totalMoves: Stores the moves till victory** **algorithm: Stores algorithms used** **difficulty: Stores AI difficulty selected**
Special Requirements	None

Property	Description
Name	playerCharacter
Description	Player Character implementation for character class
Responsibilities	Move according to player input
Methods	**setMove()** : Performs move according to player input
Attributes	**input: Stores player input** **name: Stores player name**
Special Requirements	None

Property	Description
Name	Variables
Description	Stores game/option variables
Responsibilities	Set game win condition Set game settings
Methods	**setMenu()** : Update game settings based on menu input **changeMenu()** : Change display value on options **rollDie()** : Roll game win condition, update it in game var array.
Attributes	**difficulty: Variable selecting AI difficulty** **options []: Array of option settings** **gameVar[]: Array of game variables (Win condition/Die Rolls)**
Special Requirements	None

Property	Description
Name	Board
Description	Game board that is used to play Cuadrado
Responsibilities	Draw the game board starting state Update the game board after moves Keep track of win condition
Methods	**draw()** : Draw game board **updateBoard(move)** : Change board state based on move **startBoard()** : Set starting state of the game board. **swap()** : Swap board tiles based on character input **reset()** : Reset game board
Attributes	**tile[]: Board config** **row: Rows in board** **col: Columns in board** **completed[]: Already used win conditions**
Special Requirements	None

Property	Description
Name	Buttons
Description	Implementation of the buttons on the game
Responsibilities	Check if button is pushed Perform action if button is pushed
Methods	**collide()** : Check if mouse click collided with the button **drawButton()** : Create button on screen **updateButton()** : Update button state **function()** : Runs button function upon pressing
Attributes	**state: Stores the Button state** **function: Stores the Button Function** **text: Stores the Button text** **dimensions: Stores the Button size** **value: Stores the Button value**
Special Requirements	None

6. Interface Description

See User Interface documents - upedu_ex_uiprt.pd

7. Size and Performance

The selected architecture supports the sizing and timing requirements through the implementation of the game package architecture. The client portion is implemented on the user's package. The components have been designed to ensure that minimal disk and memory requirements are needed on the client portion.

8. Quality

The software architecture supports the quality requirements, as stipulated in the Software Requirements Specification and Supplementary Specificatio