Universidade Vigo

Escola de Enxeñaría de Telecomunicación

Documentation Group C

Intelligent systems programming

CANNON DUEL

Grado en Ingeniería de Tecnologías de Telecomunicación

Aarón Riveiro Vilar

Hugo Blanco Demelo

Pablo Blanco Pumar

Renato Bedriñana Cárdenas

INDEX:

1. Description:

“Cannon Duel” is a game developed for Android using Kotlin and Jetpack Compose. The game features a battle between two players that can be controlled by the user or by an AI. The AI has three difficulty levels: random and normal. The project uses Chaquopy to integrate Python and run AI scripts.

2. Project Structure:

The project follows a standard structure of an Android application with Gradle as the build system. The main files include:

AIBehaviour.kt:

Purpose: Contains functions to handle AI behavior for both random and normal AI modes.

Functions:

- handleRandomAI: Manages decisions for a random AI.
- handleNormalAI: Manages decisions for a normal AI using Q-Learning.
- calculateShotReward: Calculates the reward for shooting.
- calculateMoveReward: Calculates the reward for moving.

Components.kt:

Purpose: Defines reusable UI components used throughout the game.

Functions:

- PlayerBar: Displays the player's health bar.
- FuelBar: Displays the player's fuel level.
- WindInfo: Displays wind direction and strength.
- GameGrid: Displays the game grid.
- InfoBox: Displays informational messages.
- AmmoSelector: Allows the player to select ammunition.
- ActionButton: A button for performing actions.
- RadioOption: A radio button option for selections.

DifficultySelectionScreen.kt:

Purpose: Provides a screen for selecting the game difficulty.

Functions:

- DifficultySelectionScreen: Displays difficulty options and handles selection.

GameLogic.kt:

Purpose: Contains the core game logic for handling actions, updating states, and running AI games.

Functions:

- handleActionButtonClick: Manages the logic for the action button.
- runUserGame: Runs a game between a user and AI.

- runAIGame: Runs a game between two AIs.
- handleShoot: Handles shooting logic.
- handleMove: Handles movement logic.
- handleNext: Handles the logic for the next turn.
- processShot: Processes a shot.
- processMove: Processes a move.
- updateWind: Updates wind direction and strength.
- updateInfoMessage: Updates the informational message.
- checkGameOver: Checks if the game is over.
- calculateHitCell: Calculates the cell hit by a shot.
- calculatePathDistance: Calculates the distance between two cells.

GamemodeSelectionScreen.kt:

Purpose: Provides a screen for selecting the game mode.

Functions:

- GamemodeSelectionScreen: Displays game mode options and handles selection

.

GameOverScreen.kt:

Purpose: This file displays the game-over screen with the final results. It also has a button to return to the main menu.

Functions:

- GameOverScreen: This screen displays each player's remaining health points and has a button at the bottom so you can return to the main menu.

GameScreen.kt:

Purpose: Provides the main game screen where the game is played.

Functions:

- GameScreen: Displays the game screen and handles game logic based on the selected mode.

MainActivity.kt

Purpose: The main entry point of the application, initializes the app and manages navigation.

Functions:

- onCreate: Initializes Python and sets up the main content.
- ManageNavigation: Manages navigation between different screens.

TrainingScreen.kt:

Purpose: Provides a screen for selecting the number of training games.

Functions:

- TrainingScreen: Displays a slider for selecting the number of training games and handles selection.

3. Game Flow:

1. Application startup: The application starts in **MainActivity**, which sets the content using **CannonDuelTheme** and calls **ManageNavigation**. Python is initialized, and the AI module is loaded.

2. Navigation between screens: ManageNavigation manages navigation between the player selection, difficulty selection, and game screen (**GameScreen**) screens.

3. Player and difficulty selection: The user selects the player type in **PlayerSelectionScreen** and the difficulty in **DifficultySelectionScreen**. Clicking “Next” navigates to the next screen.

4. GameScreen: In **GameScreen**, the user interacts with the grid to select cells and perform actions such as shooting or moving.
The game logic in **GameLogic** handles the user's actions, updating the game state and providing feedback through the user interface. If we are using an intelligent plater

5. Game state update: Functions in **GameLogic** update the game state based on user actions, such as shooting and moving, and update the user interface accordingly.

4. AI Implementation:

We have three difficulty levels: Easy, Medium, and Hard. AIBehaviour.kt is the file that manages the intelligent players and their respective functions. We use a Python script for the smart players to use reinforcement learning. Medium and Hard mode use Q-Learning to make decisions, using Chaquopy to call the Python functions.
- Easy mode:
In this mode, you play against an agent that plays randomly. It shoots when it has ammo at a random cell. Then, it shuffles all the available cells where you can move, goes through the list of cells, and chooses the first one it finds.
- Normal mode:
In this mode, we use reinforcement learning, specifically Q-learning.
We have two Q-Tables, one for movement and the other for shooting. The q table for movement is a three-dimensional matrix, and the one for shooting is four-dimensional.
In the very first game, both matrices are initialized with zeros, and they will update their values according to the rewards obtained after each iteration. Then, the matrices are stored in a .npy file, so in the next games, the agents will not start with zero matrices.

5. Final UI Implementation