Game Dev Report - 19300753 Ruth Dooley

This game development project uses Severus, a javascript 3D engine and Node.js. This project is developed using Robotify (the company I work with) software.

Theme and Narrative:

This is a game designed to help people learn the basic rules of chess. The user controls the yellow drone (Fig. 1) by giving it orders to interact with the chess pieces, which it fulfils.

Education / AI Co Learning

There are 5 different levels that demonstrate different chess piece's (pawn, bishop, knight, rook, queen) valid moves under different circumstances. Each piece has a demonstration by the AI Co-learner to display the piece's rules (location on board and valid moves). The game involves solving puzzles for each of these five categories using the rules demonstrated, where a random puzzle for that piece (pre-determined) is chosen. The overall quiz involves testing the user with a random puzzle for a random piece for the overall quiz.

The user gives the main avatar drone instructions to move a piece to a valid square. If the move is invalid the AI Co-learner will demo a valid move and then explain the piece's valid moves.

Game / Physics

This game integrates collisions (between the pieces and the ground) and gravity (pieces falling after being dropped). The advanced physics feature implemented in this project is improving upon the current engine particle system (only facilitates linear motion). This project implements a parabolic particle system (Fig. 2), used to create a confetti effect displayed on quiz completion.

Devices HCI

Hosted using the Severus engine on a laptop the user can control the movement using a game controller which once connected to the device, can communicate with the project. This can also be controlled with the w, a, s, d (directions), e (enter), b (back) keys, if a controller is not connected. These features not explicitly defined (on par with Unity complexity), needed to be inputted manually. Alternatively the game can be played on a mobile device using python code development. Python commands can be used in the Robotify IDE (Fig. 3). This included creating the Python wrapper functions.

Features

Mentioned Above

- External controller implementation (Controller.js). This is with the support of Robotify functions to connect controllers, however necessary configuration needed for keyboard to button mapping.
- 5 different lessons included in this project, where 2 was minimum.
- Demo levels created to explain the basics to the users.

Sounds (Sound.js)

Sounds for the piece pickUp and dropOff and robot speaking are included.

Text Sprites

Text sprites (Fig. 4) attached to the drone, in order for the AI Co-learner to "talk" to the user.

Piece Move Generation

The intention of the original design was to implement an exhaustive chess ruleset (this became far too complex, scope for project reduced). The piece movement for each piece type is detailed

in the respective class. Although not implemented in the game, king in check validation (King.js), en passant (Pawn.js) and castling (Rook.js) all integrated.

GUI (Gui.js)
(Fig. 5) Used to determine the type of game mode (Demo or Quiz) and the piece type.

Visual Cues
Last move display (red square (Fig. 6.1)) displays the last moved piece start and end position. Visualising valid moves (small grey squares (Fig. 6.2)) used to visualise the valid moves that can be made by that piece. Controller position (blue square (Fig. 6.3)) represents the user's controller position (necessary and not really an added feature). These features enhance the user experience and HCI.

Animations (Drone.js)
The drone has a number of animations (idle, carry, drop, victory, lose), all which are updated using the update loop.

Camera view toggling (env.js)
Three camera perspectives, board from top, board from angle, and drone face on view.

Lesson and Demo Adding (Game.js)
Easily facilitating new lesson and demo creation by adding another case in the relevant switch. This is facilitated by the manual board piece initialization where the board setup is easily editable (env.js).

Design, Documentation, Version Control
Clean cod design (OOP & inheritance), supplementary comments describing functions and variables and necessary git version control.

Future Works

- Gravity is not predictable, the pieces falling and sometimes falling over and sometimes not. Would love to have the time to address this however would be outside the scope of the time needed to spend completing this assignment.
- Python scripts don't really make a lot of sense at the moment, as it is only one command but this is with the intention of expanding upon, where chess openings could be taught or alternatively Python commands could be taught using the game of chess.
- Fully implement the rules of chess and valid movement generation. As mentioned above tried this (spent 2 full days on it) however kept getting stuck in this loop. Getting piece valid moves > does moving this piece put its own king in check validation by getting other pieces valid moves.
- Tried manually initialising the board giving board notation from real games (https://www.365chess.com/) however this heavily relied on the condition that all of the piece valid moves were airtight (which was not possible given time frame).

References

3D Assets and Sound: Robotify https://www.robotify.com/
Existing Controller Connection: Robotify https://www.robotify.com/
Linear Particle System: Robotify https://www.robotify.com/
Square notation: https://www.cgtrader.com/free-3d-models/various/various-models/

Supplementary Screen Captures

Note* Screen capture taken with temp chess piece assets.

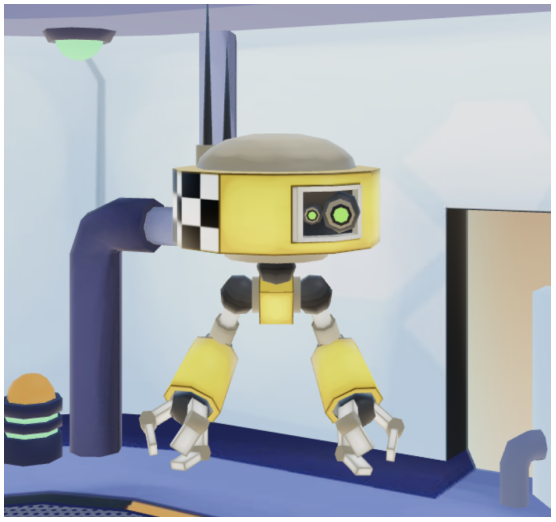


Figure 1

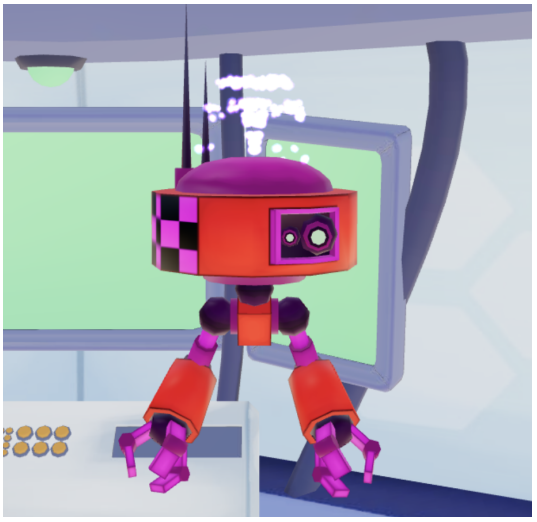


Figure 2

Figure 3

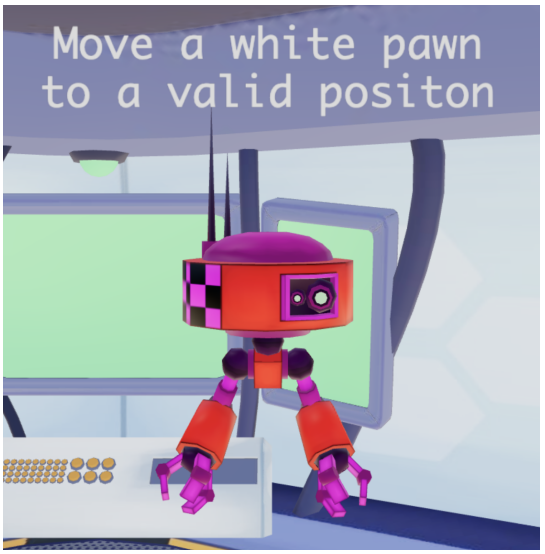


Figure 4

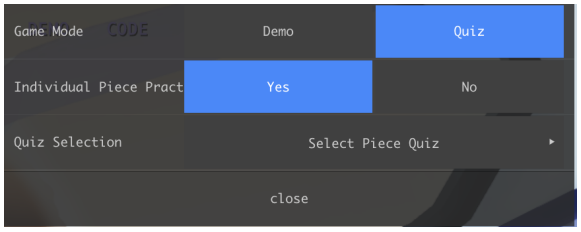


Figure 5

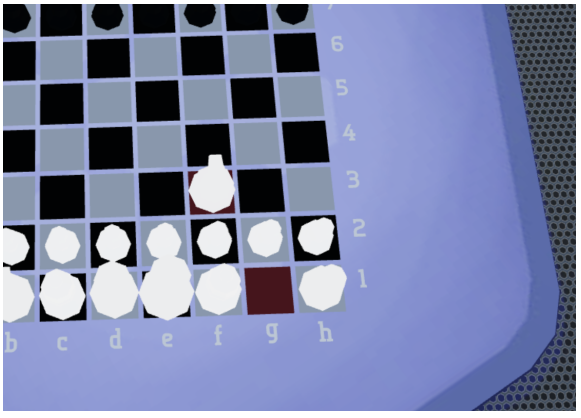


Figure 6.1

Figure 6.2

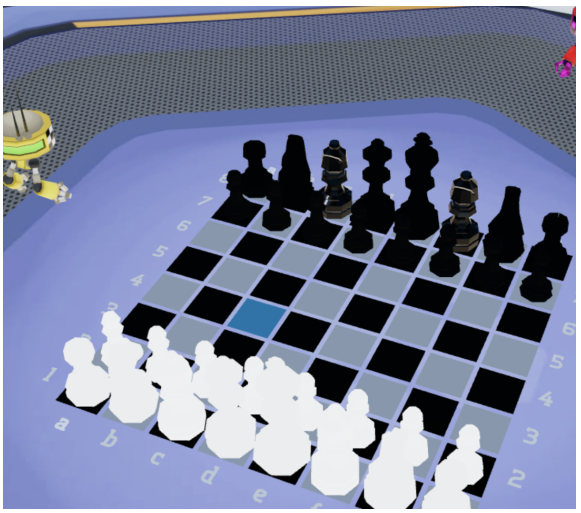
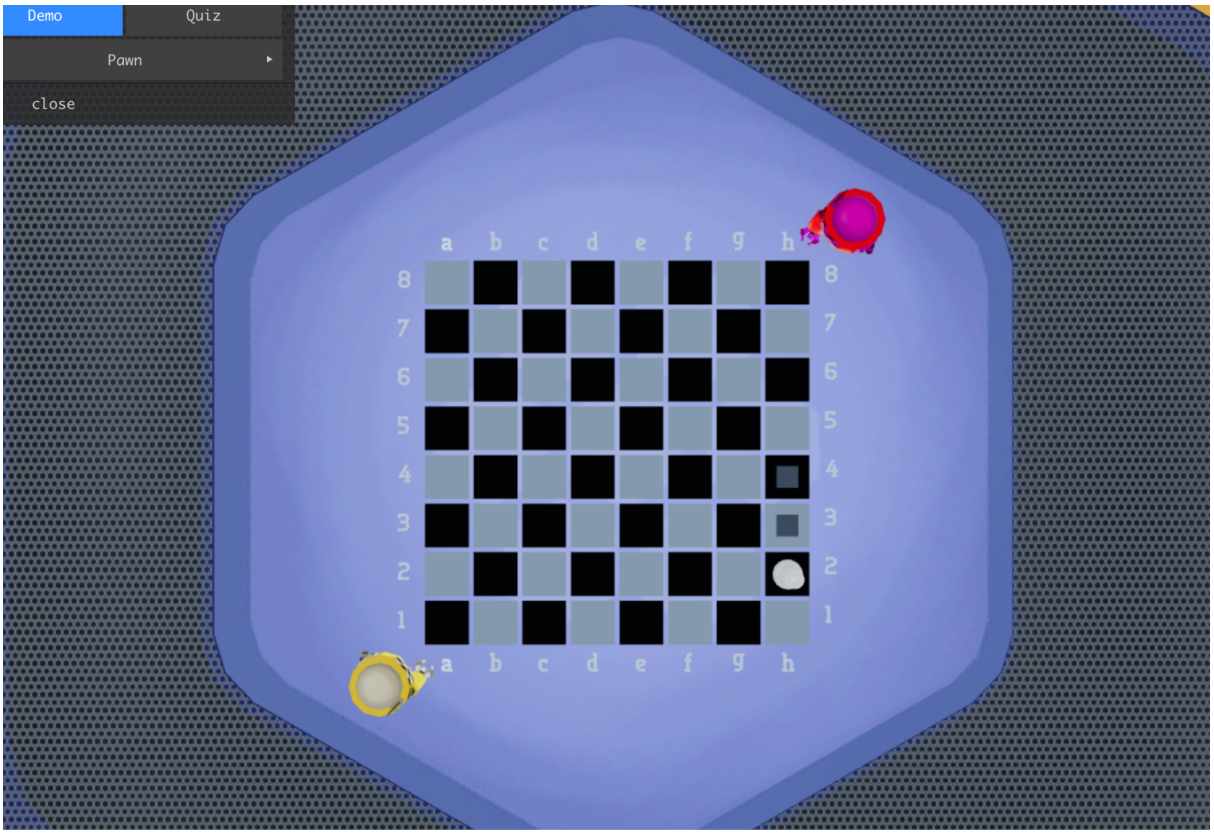


Figure 6.3

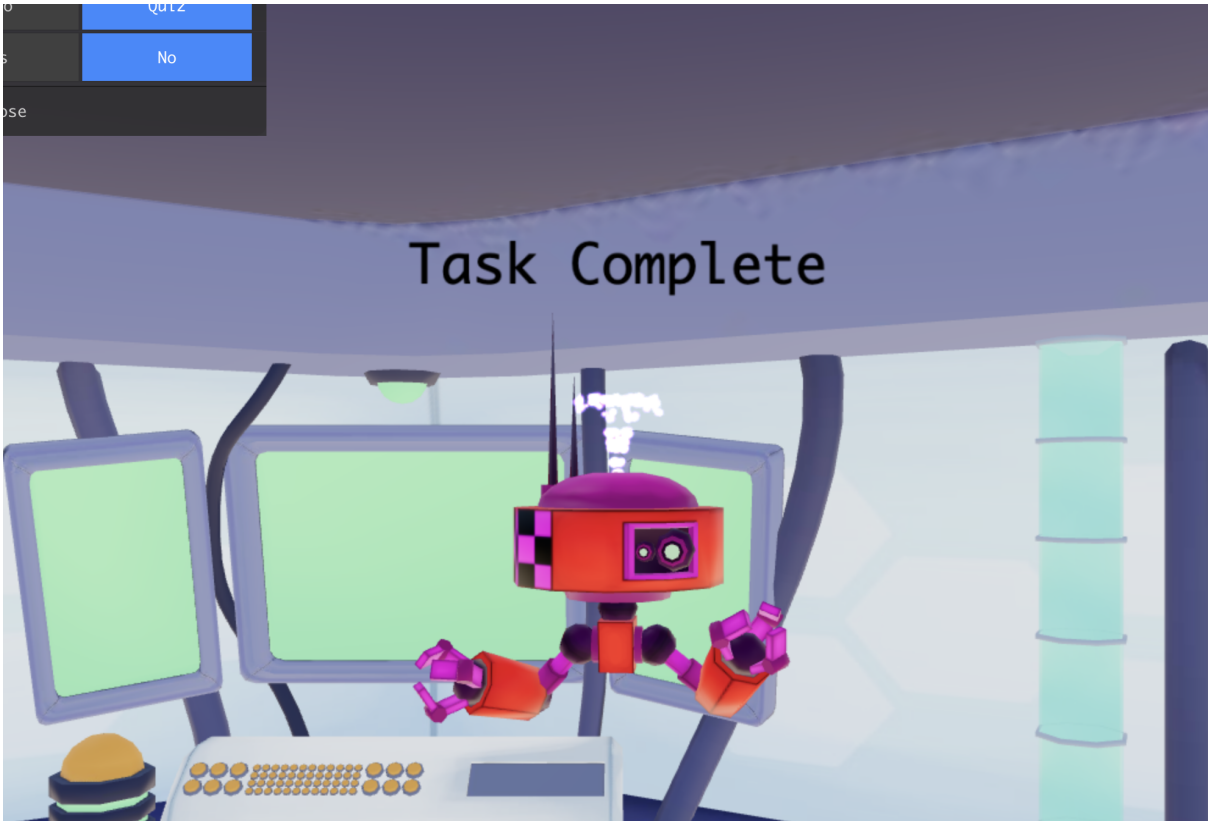
Game Play Screen Captures




Demo
Quiz
Pawn
close


No


Quiz
No


Quiz
No
Task Complete