4/8/2024

GADE 7321

Part 2

Kegan Wade Bennette (ST10074145)
Martin Louw (ST10208767)

Table of Contents

Figure List:

Part 2 Document:

High Concept Statement:

For our game we are doing an Advanced Connect Four type game.it is a strategic two-player game that combines the classic gameplay of Connect Four with exciting power-ups and a larger playing field. The game is turn-based, non-random, and provides perfect information to both players.

Game Rules:

The

- The game is played on an 12x12 grid.

- Players take turns dropping their coloured puck, either blue or red, into any non-full column of the grid.

- The objective is to connect five of one's own discs of the same colour next to each other vertically, horizontally, or diagonally, before your opponent. Use your mouse pointer to choose where you want a puck to land.

- There are special 'power-ups' that can be used. When a player uses a power up by pushing 1,2 or 3 on their keyboard, they get a special one-time ability such as removing an entire row, removing an entire column or taking an extra turn. Each of these power ups can only be used once per player, per game.

- The game is over when the first player to get 5 discs in a row wins.

Power Up Abilities:

- Remove a row: The player that uses this power up chooses to remove an entire rows discs.
- Remove a column: The player that uses this power up chooses to remove an entire columns discs.
- Extra turn: Gives the player that uses this power up an extra turn.

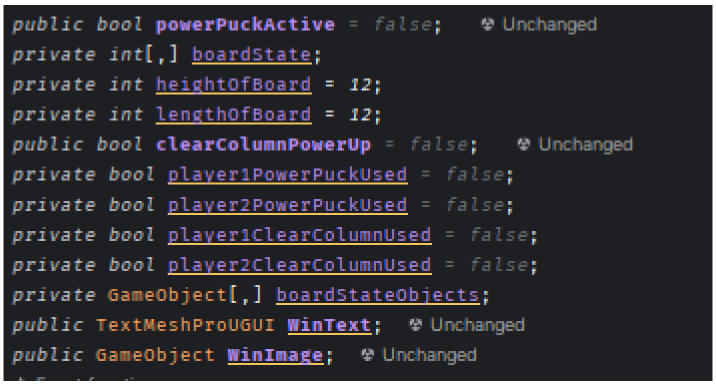
Game State Representation:

The game state is represented as a 12x12 double array (boardState[,]), where each cell corresponds to a slot in the grid on our unity board. Each cell can be empty, filled with player 1's puck or filled with player 2's puck.

This also works using unity's built in collider system. We used a double array to store the game state of the board. The board will then be updated via an UpdateBoardState() method to change and store which colour of disc is sitting in the array in the following turns. This update method then updates the current boardState[,] method to its new state.

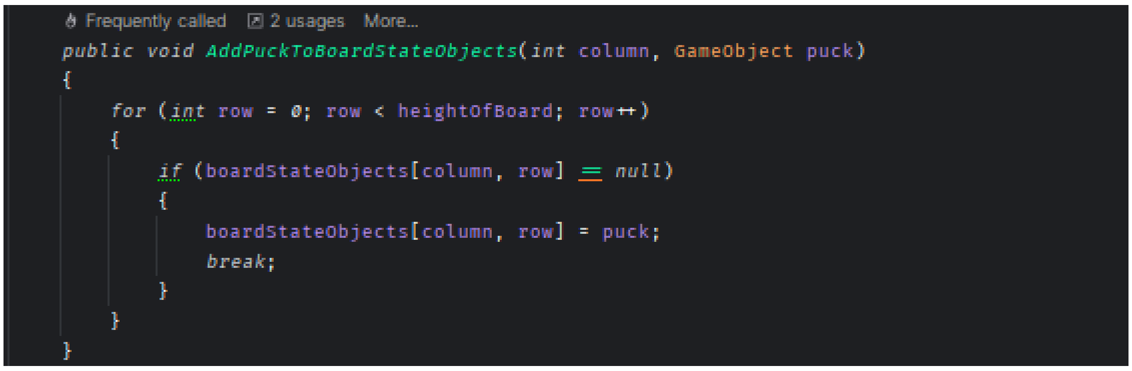
Our AddPuckToBoardStateObjects() method takes in the column the power up is being played and adds the object to the boardStateObjects[,] array.

We also have an IsColumnFull() method, this method checks in which column you played in and then checks if the column is already full and wont allow you to play there.



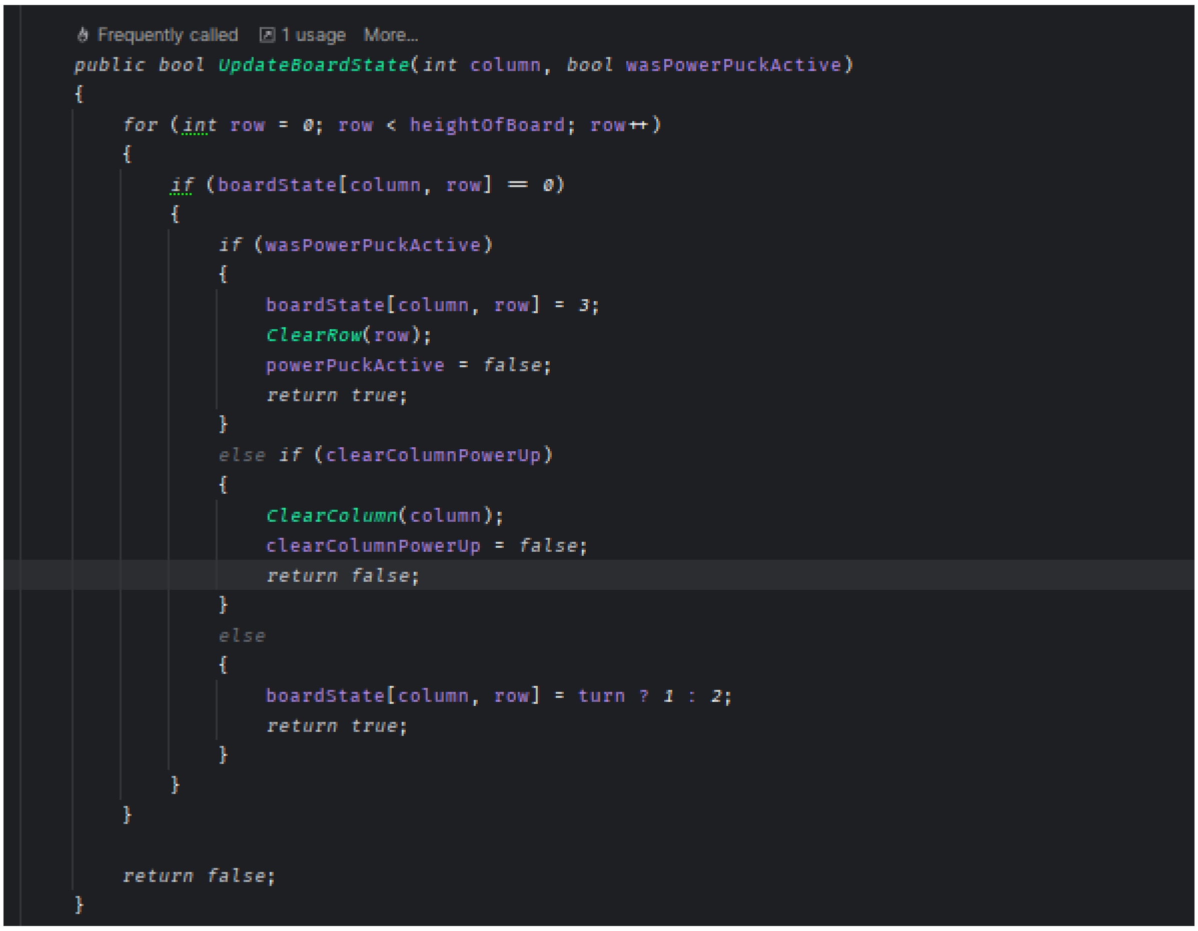
```
public bool powerPuckActive = false;    Unchanged
private int[,] boardState;
private int heightOfBoard = 12;
private int lengthOfBoard = 12;
public bool clearColumnPowerUp = false;    Unchanged
private bool player1PowerPuckUsed = false;
private bool player2PowerPuckUsed = false;
private bool player1ClearColumnUsed = false;
private bool player2ClearColumnUsed = false;
private GameObject[,] boardStateObjects;
public TextMeshProUGUI WinText;    Unchanged
public GameObject WinImage;    Unchanged
```

Figure 1: Initializing (Louw, 2024)



```
    Frequently called   2 usages   More...
    public void AddPuckToBoardStateObjects(int column, GameObject puck)
    {
        for (int row = 0; row < heightOfBoard; row++)
        {
            if (boardStateObjects[column, row] == null)
            {
                boardStateObjects[column, row] = puck;
                break;
            }
        }
    }
```

Figure 2: AddingPucks (Louw, 2024)

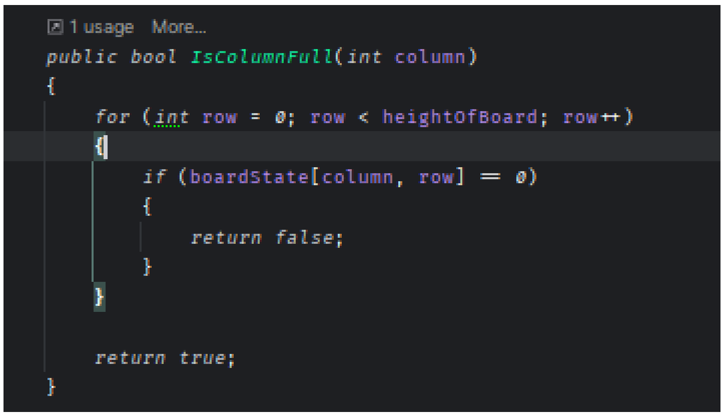


Frequently called · 1 usage · More...

```
public bool UpdateBoardState(int column, bool wasPowerPuckActive)
{
    for (int row = 0; row < heightOfBoard; row++)
    {
        if (boardState[column, row] == 0)
        {
            if (wasPowerPuckActive)
            {
                boardState[column, row] = 3;
                ClearRow(row);
                powerPuckActive = false;
                return true;
            }
            else if (clearColumnPowerUp)
            {
                ClearColumn(column);
                clearColumnPowerUp = false;
                return false;
            }
            else
            {
                boardState[column, row] = turn ? 1 : 2;
                return true;
            }
        }
    }

    return false;
}
```

Figure 3: Update BoardState (Louw, 2024)



1 usage · More...

```
public bool IsColumnFull(int column)
{
    for (int row = 0; row < heightOfBoard; row++)
    {
        if (boardState[column, row] == 0)
        {
            return false;
        }
    }

    return true;
}
```

Figure 4: Column Full Check (Louw, 2024)

Utility Function:

Player Advantage:

To evaluate the advantage of each player, we can consider the following factors:

1. Potential Winning Positions: We will go through the board and find possible winning positions for every player. They can be in horizontal, vertical or diagonal lines where a player has four or more discs together. We'll count the number of discs belonging to the player in these positions.

2. Count the Discs: We are going to count all the discs of each player that are on the board.

Winning Moves:

We will look to see if any of the players can make a winning move on the board. This checking process includes searching for sequences that have four pucks from only one player in a row, column or diagonal direction.

Power-ups:

In case power-ups are part of it, we should think over how they can change the condition of the game state. Let's say someone deletes a row or column by using a power-up, this action might modify the board state. It could influence who has more advantage among players and also alter possible winning moves for different players in this match.

Utility Score:

- AdvantagePlayer represents the advantage of the player calculated based on disc counts, potential winning positions. AP = (Disc counts + Potential win)
- WinningMoves indicates whether either player has a winning move on the current board.
- Power-upImpact evaluates the potential impact of power-ups on the game state.
- ($w1 = 1$), ($w2 = 2$), and ($w3 = 3$) are weights assigned to player advantage, opponent disadvantage, and winning moves respectively.

Example:

Utility Score = $w1$·AdvantagePlayer+$w2$·Winning Moves+$w3$·Power-up Impact

Utility Score = $w1$·(1 + 3) +$w2$·(2) +$w3$·(1)

Utility Score = (1)·(1 + 3) +(2)·(2) +(3) ·(1)

Utility Score = 11

Github Link:

https://github.com/MartinGLouw/GADE7321_Part2.git

References:

Louw, M., 2024. *Adding Pucks*. [Online]
[Accessed 28 April 2024].

Louw, M., 2024. *Column Full Check*. [Online]
[Accessed 28 April 2024].

Louw, M., 2024. *Initializing*. [Online]
[Accessed 28 April 2024].

Louw, M., 2024. *Update BoardState*. [Online]
[Accessed 28 April 2024].

Kegan, B. 2023. P1DeleteCol. [Personal Drawing]. Cape Town: Unpublished.

Kegan, B. 2023. P1DeleteRow. [Personal Drawing]. Cape Town: Unpublished.

Kegan, B. 2023. P1Skip. [Personal Drawing]. Cape Town: Unpublished.

Kegan, B. 2023. P2DeleteCol. [Personal Drawing]. Cape Town: Unpublished.

Kegan, B. 2023. P2DeleteRow. [Personal Drawing]. Cape Town: Unpublished.

Kegan, B. 2023. P2Skip. [Personal Drawing]. Cape Town: Unpublished.