# GADE 7321 Concept Document

Francesca Fitzgerald ST10143178

## High Concept Statement

Crown Defense is a digital strategy board game where two players must cleverly use their board pieces to defend their crown while working to capture their opponent’s crown.

## Game Rules



Figure 1: Crown Piece (Blender, 2024)

The game is a two-player game set on an 8x8 checkered-tile board.

**Basic Rules:**

* The game is a turn-based game.
* When their turn begins, a player may move ONE piece on the board.
* A turn ends when the player has moved a piece.
* Each player has 13 game pieces:
  + 4 guards,



Figure 2: Guard Piece (Blender, 2024)

* + 5 mercenaries,
  + 3 shields, and
  + 1 crown.
* A guard can move linearly along any number of tiles.
* A mercenary can move diagonally along up to a maximum of 3 tiles.
* A shield can move linearly along up to a maximum of 2 adjacent tiles, or diagonally along 2 tiles.
* The crown can only move to 1 adjacent block (not diagonally).
* Each player must attempt to capture their opponent’s pieces by moving to the tile behind the target piece.



Figure 3: Shield Piece (Blender, 2024)

* If the tile behind the target piece is occupied, the piece cannot be captured.
* If a piece is moved beyond the tile behind an enemy piece in the same turn, this does NOT capture the enemy piece.
* Players can move their game pieces over any other allied game pieces (except the Crown) to traverse the board.
* The game ends when a player captures the enemy crown.



Figure 4: Mercenary Piece (Blender, 2024)

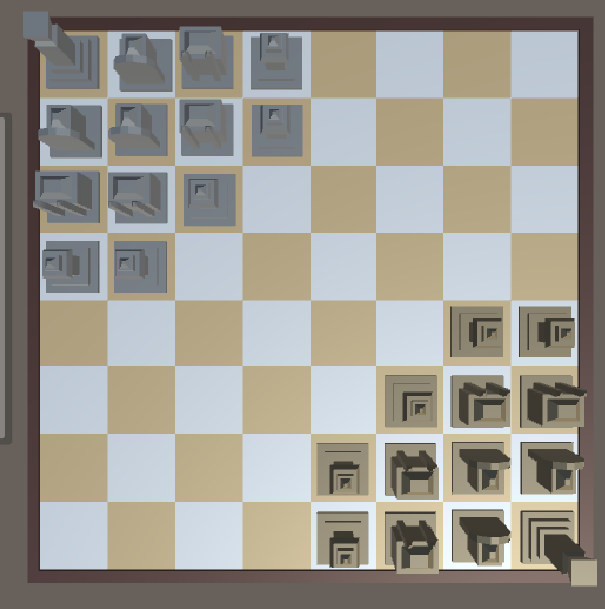


Figure 5: Starting Board (Unity, 2024)

**Board Layout and Set-up:**

At game start, the game board should always appear like the layout in figure 5 (right).

The game will always start on the gold player’s turn and the grey player will always play second.

The game is played from a top-down perspective to allow the players to both have a complete view of the board.

## Game State Representation

**The Game Pieces:**

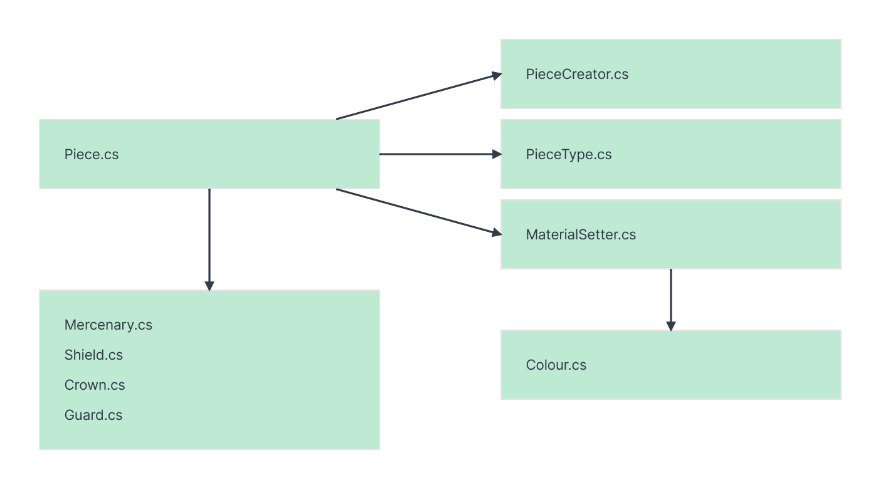


Figure 6: Game State Diagram 1 (Fitzgerald, 2024)

Each piece will be constructed from the Piece class, inheriting the standard characteristics and base movement methods.

The Piece Creator class will use a dictionary to store each piece prefab, using the dictionary to instantiate the appropriate piece where needed.

The dictionary will use the Piece Type enumerator class to instantiate the correct piece along with the correct prefab, colour, and movement abilities. Each piece class will have the necessary movement mechanics for the corresponding piece type.

The Material Setter class will be used when a piece is initialized, using the Colour enumerator class to set the correct material.

**The Board Layout & Board Information:**

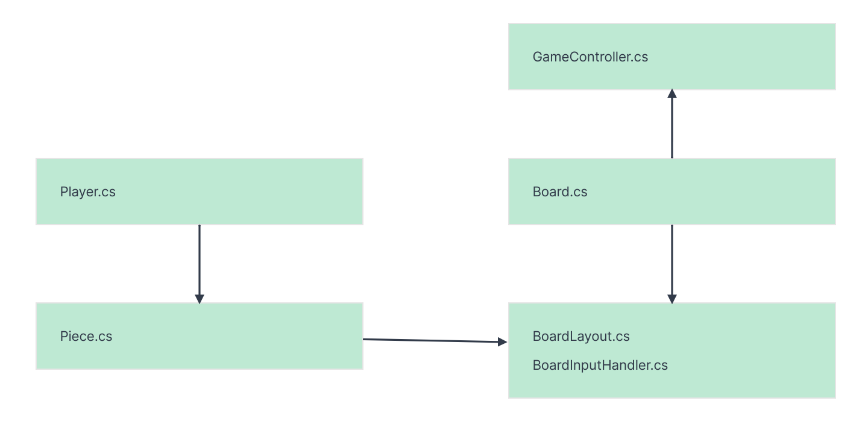


Figure 7: Game State Diagram 2 (Fitzgerald, 2024)

The game uses a scriptable object to apply a pre-determined board layout. The Game Controller class initializes the game, applying the layout and using the Piece Creator class to create all the needed pieces for the board layout.

The Board Input Handler class is used to process the information from the square the player has clicked on when moving a piece.

The Board class processes all coordinate information relating to the pieces and board squares. This class will check where the pieces are, and where they can move to. This class will make use of a 2D-array to store the positions of each piece on the board, making it easy to check if a square is occupied. This class makes use of many methods to calculate the required coordinate information to move the pieces in the board.

**Game Information & Game State:**

The Game Controller class does the majority of the communication between the other classes and game objects.

The Game Controller at game start initializes the two players and the board. It will manage the players’ turns and what pieces can be controlled during the respective turns. This class will also use methods to generate the available moves for each player.

The input receiver classes are interfaces used to communicate the input from the board to the Game Controller, to be used to calculate moves.

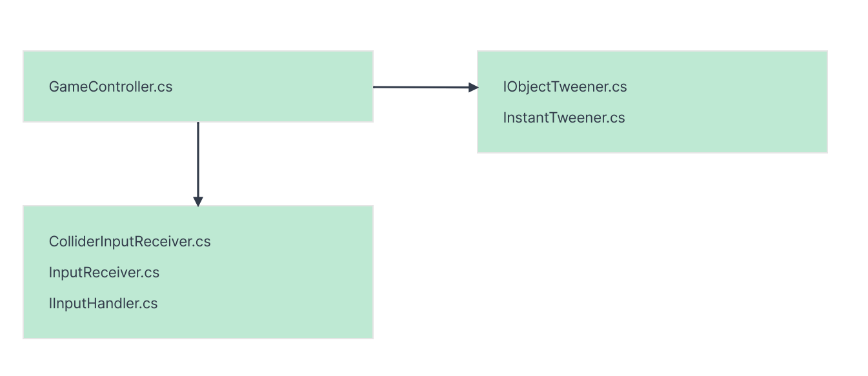


Figure 8: Game State Diagram 3 (Fitzgerlad, 2024)

The Collider Input Receiver class uses Unity’s physics raycast method to detect what square the player is clicking on. This is used to then calculate if the selected piece can move to that square.

The I Object Tweener and Instant Tweener classes are used to animate the movement of the pieces when they are moved on the board. This is just an easy way of animating the pieces instead of creating the animations in Unity.

## Utility Function

The utility function is the “training signal” of a machine learning algorithm by checking the “goodness” of the outcomes generated by the AI. The utility function will be used to measure the chance of success of a specific decision path, which the AI will use to choose its moves.

**Utility Function Equation:**

*Where:*

= the likelihood of a win if the game state is chosen

= the game state number

= the expected utility

**Explanation:**

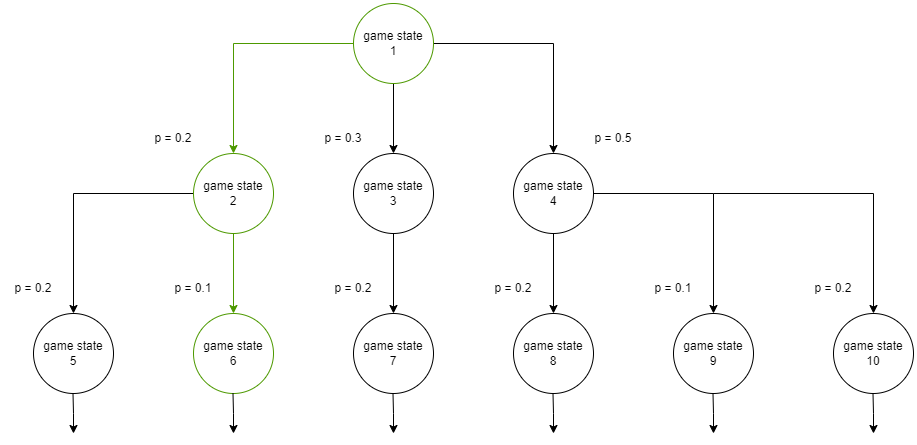


Figure 9: Behaviour Tree Diagram (Fitzgerald, 2024)

In the diagram above (figure 9), we can see that the game’s states are represented in a behaviour tree. This is an easy way of representing how the game states can be chosen to ensure an optimal outcome (ie: a win).

The “p” in the diagram represents the probability of a win if that game state is chosen. If the values are added horizontally, they should equal 1. If the probability corresponding to the game state is 1, that means there is a 100% chance of a win, and a probability of 0.3 means a 30% chance of a win, etc. etc.

When a path is chosen in the BTD (behaviour tree diagram), the probabilities are not added but multiplied to get the outcome of that decision path. For example, if the green path is chosen, we can calculate the probability of a win by multiplying all the p values:

*[equation 1]*

Compare this to the probability of another path:

*[equation 2]*

We can see that equation 2 has a probability of 10%, a much higher probability than equation 1 which is 2%. This will form the basis for how we will evaluate the “goodness” of the chosen game states.

The utility function will use this calculated probability to calculate the expected utility of a chosen decision path on the BTD – the higher the probability, the lower the expected utility. This expected utility () must be as low as possible to guarantee a win – ie: the algorithm(s) must aim to minimize the utility function.

The game can take two courses of action:

1. calculate the p-values of the game states ahead of time to allow the AI algorithm to calculate the utility function and choose the most optimal path (using a *minimax* algorithm).

OR

1. calculate the p-values as the game evolves and game states are discovered, calculating the expected utility from the available values (using a *monte carlo* algorithm).

This will be decided in the next part of this project.