Non-Exam Assessment

--Analysis for the NEA project--

This project consists of the creation of an computer-based version of Love Letters.

A friend of mine, Douglas Tennant, is a keen fan of board games, including the game Love Letters. He wishes to play love letters with others online or by himself against some virtual opponents.

Overview:

Love letters is a game where multiple people compete for the heart of a princess.

The game revolves around a set of cards depicting characters found in the palace:

The Guards:

-Worth only 1 point, there are five of these cards. When played, the player can accuse someone of holding a certain card. If the accusing player guesses the correct card, the challenged player goes out of that round. However, a guard cannot accuse another player of being a guard.

The priests:

-Worth 2 points, there are 2 priest cards. When played, the player can choose one other player to reveal their card to them only.

The barons:

-Worth 3 points, there are 2 barons. When played, the player and one other player enter a challenge. The highest ranked card wins and the loser goes out.

The handmaids:

-There are 2 handmaids, both worth 4 points. When played, the player is protected until their next turn. Others cannot target them with effects from playing cards.

The princes:

-Princes are worth 5 points, and there are two of them in the kingdom. When played, the player chooses one other player to duel and they must discard their card.

The king:

-The king is worth 6 points. Playing him allows the player to swap cards with another person.

The countess:

-The countess is worth 7 points. The countess has no special ability once played, but must be discarded if the player has both the countess and either the king or the prince.

The princess:

-Worth the most out of all the cards, the princess is worth 8 points. If the player discard the princess through any means, they must go out.

Each round begins with one card removed from the pack and one card given to each of the players.

Diagram

Description automatically generated

A random player goes first (the winning player in later rounds) and draws a second card. One of the two cards must be played, often causing a special effect. The card that was not played is the one used in any effects that involve the player. The played card is placed in front of them for others to see.

Text

Description automatically generated

The player can deduce which cards other players may have and choose what to play accordingly. For example, if the player knows the opponent has a certain card they can accuse the opponent of being that card with a guard, challenge them with a baron to make them go out. They may, however, decide to use a low ranked card at a time when they won’t necessarily help in order to keep their higher ranked card.

Diagram

Description automatically generated

Numerous events may end in a player going out. In this case they must discard their card without its ability going into effect and won't have any subsequent turns in the round.

When all cards have run out, unless only one player is left, the values of the cards are compared and the players with the lowest scores go out.

The remaining player wins one heart from the princess.

The hosting player may choose any amount from 1 - 10 hearts needed to win the game. This would be decided when the game is started by the host. The rounds will repeat until a player has reached the specified necessary amount of hearts.

Specifications:

The program must have a way to specify how many players will be playing (2 - 4) and whether the opponents are player connected from another computer or if they are a computer opponent.

The computer based opponents must have a difficulty setting to make it easier or harder for the players to beat.

The hardest opponents should be able to predict what cards other players have.

The easiest opponent should make small errors and be more erratic.

The cards must all have a value and an ability that happens when it is played or some other effect that may come into play.

All players must have a name and an in/out state.

Each player should have a counter of how many hearts they have won in total.

For every player there should be previously played cards and a current card (that can only be seen by them and actions that involve the card's role).

The system must be friendly to all users, supplying short descriptions of what every button does, and providing a tutorial. This to prevent players getting lost or having difficulty playing the game.

The tutorial must guide new players that are learning from scratch, but also help returning players remember how to play.

There must be a suitable main menu containing a host and a join button, letting players create their own game with a combination of computer players or online players, or join an existing game once given an IP and port.

--Documented Design--

Firstly, an important note about Unity.

Sometimes to refer to methods in classes, that have been separated into different scripts, it is necessary to refer to them by their GameObject first. This is because Unity has everything split into GameObjects in a main Scene. This behaves the same as scripts in a project in a Visual Studio .NET console program just with a few extra lines of set up.

The first code implemented is the code for the cards which the game revolves about.

The initial plan was to create a card GameObject and a behaviour script that would communicate with other scripts, but turned out too complex and not the best solution.

A namespace called manager was created to put classes, methods and functions in and used by a central game script that will create instances of objects and process the event system.

A main Card class was made with its value, name, type and description that will be used by the game. It has three virtual methods that should occur once it has been drawn, when it is played and when it should be discarded. 8 subclasses inherit Card’s variables and override the necessary virtual methods based on the effect that the card should have.

First the GuardCard is given value <- 1 and should override the Play() function to:

Function Play (Player challenger)

Set Player challenged to an opponent chosen by the challenger

Set string ‘guess’ to a type of card chosen by the challenger

If the challenged’s card type matches the string ‘guess’ then

Tell the challenger they were correct

Take out the challenged player

Else

Tell the challenger they were wrong

End the If statement

End of Function

The PriestCard is given value <- 2 and should override the Play() function to:

Function Play (Player challenger)

Set Player challenged to an opponent chosen by the challenger

Tell challenger the opponent’s card type

End of Function

The BaronCard is given value <- 3 and overrides the Play() function to:

Function Play (Player challenger)

Set Player challenged to an opponent chosen by the challenger

If challenger’s card’s value is greater than that of challenged then

Take out the challenged player

Else the challenger’s card’s values is less than that of challenged then

Take out the challenger

Else

Tell both players that they have the same type of card

End od if statement

End of Function