Project 1

CIS-17C

Heidi Dye

10/10/19

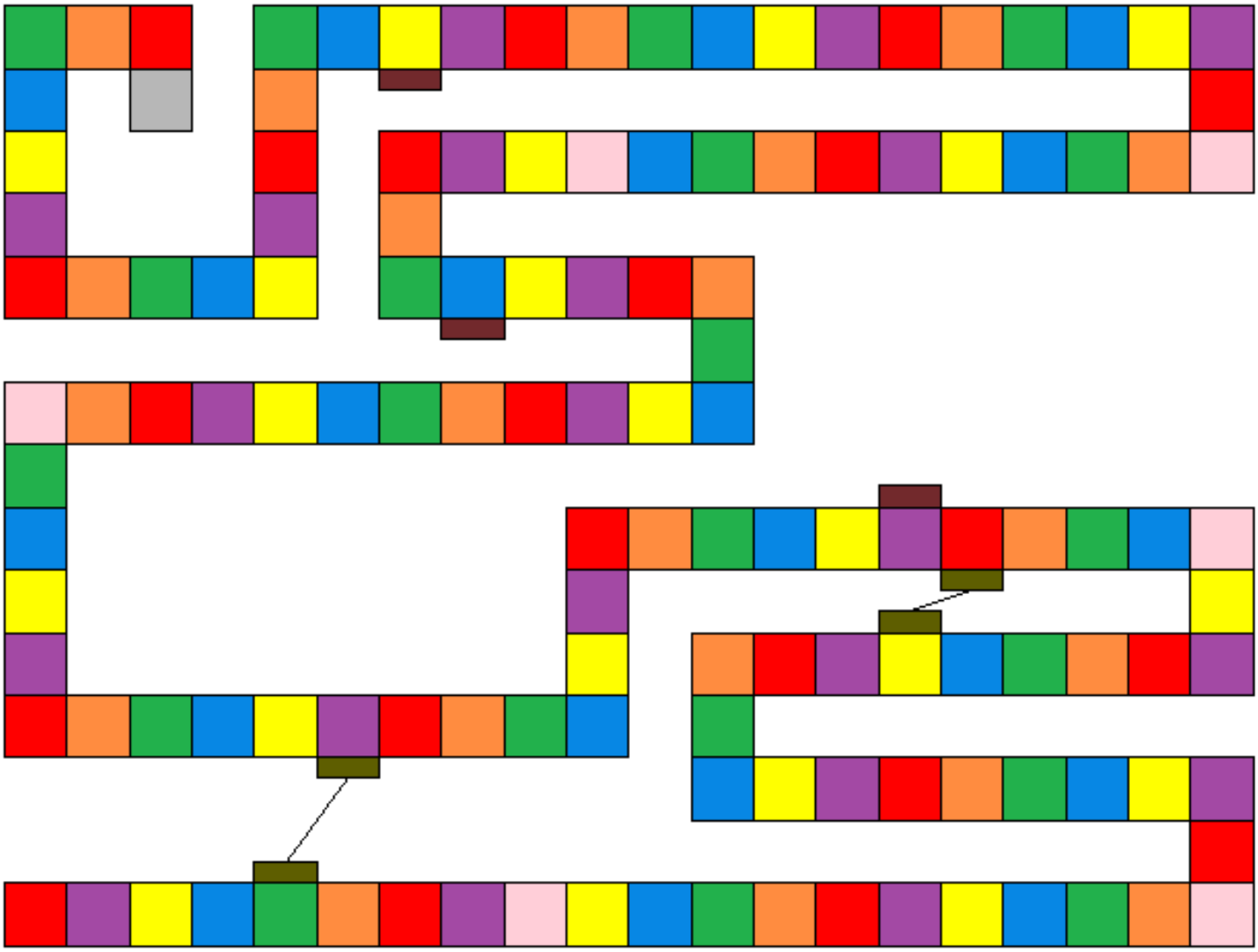
Introduction

Title: *Candyland*

The Game of Candyland is a board game that is played with 2-4 players. The object of the game is to be the first player to reach King Candy's castle. There are four different colors; red, blue, yellow, or green that a player can be that is represented by a gingerbread man of the chosen color. The game board has a track of 134 colors that lead to the castle. During a player's turn, a player chooses a card and moves their gingerbread man based on the color on the card. Cards have the colors; red, purple, blue, yellow, orange, and green, and they can be either double or single. If a player chooses a specialty character card, then they can move their gingerbread man to the respective character spot on the board even if the character takes them back on the board.

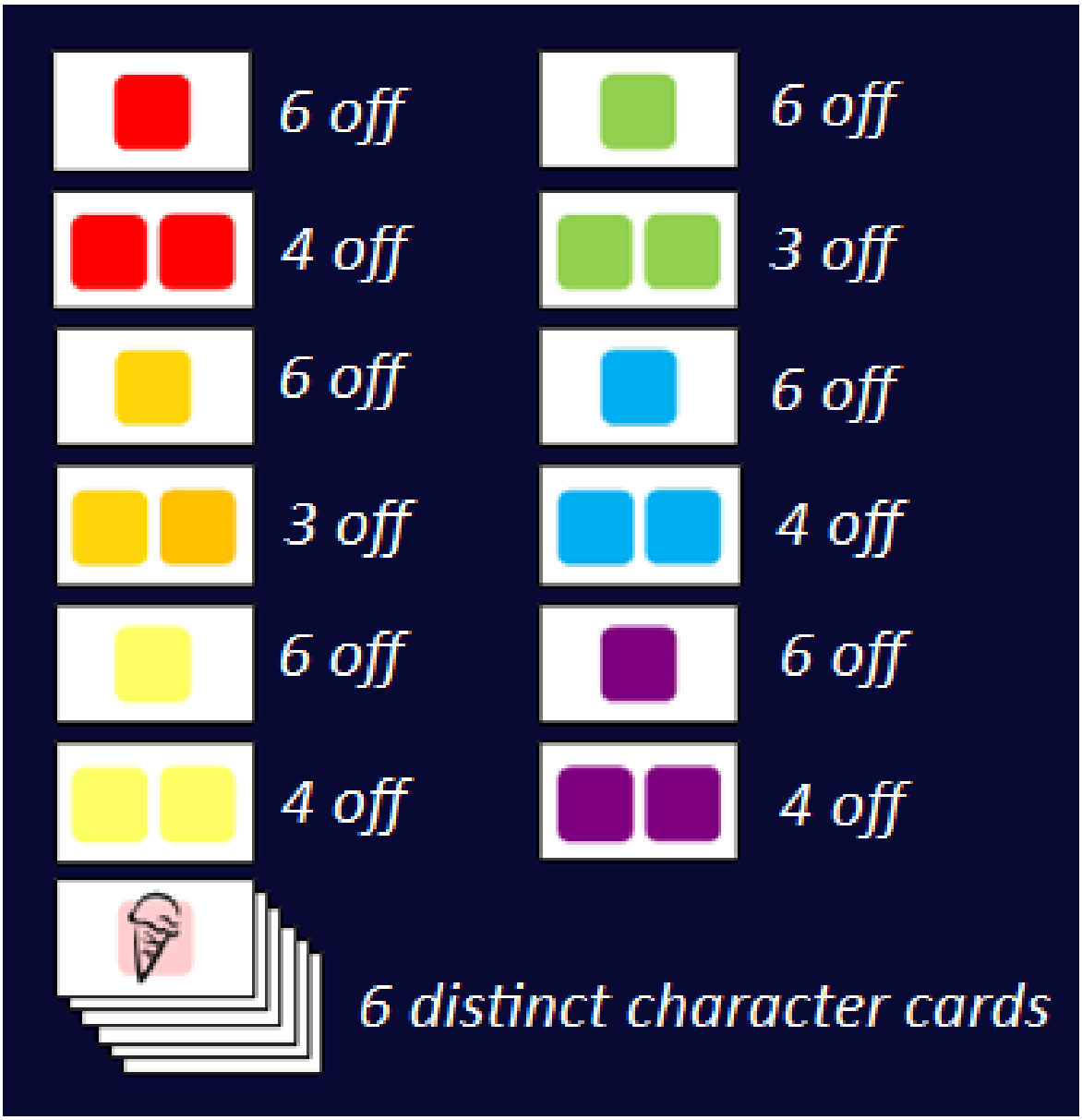
Board

This is what that Candyland board looks like, or at least what the version from my childhood looks like.



I will be using this board that I found on *Data Genetics* as a reference to make things simpler on my part

Cards



Once again, I found this on Data Genetics. For my distinct character cards, I chose: “Frostline”, “Mr. Mint”, “Gramma Nut”, “Jolly”, “Plumpy”, and “Princess Lolly.”

Summary

Project Size: 750+ lines

Number of Classes: 4

Number of Variable: 29

Midterm Concepts Used: Maps, Pairs, Iterators, Queues, and Algorithms

I finished this program in two days in roughly sixteen hours.

I tried to implement stacks, but I felt like they did nothing to help the program, so I took opted out of using it. The game ended up being pretty much self-automated to the point where there was no need for a player. So I added in some user input to make it feel like the player is actually playing and drawing their own cards. Most of the problems that I faced was due to stupid error on my part. Even though Candyland is relatively an easy project in theory, there is a lot of logic to consider when coding. In the end, I think that I figured out the logic, but this logic did end up making the game self-automating. Whether that is a good or a bad thing I have no clue.

Psuedocode

Get the number of players from the user

Initialize a game

Make a deck of cards and shuffle it

Set up the board game with the colors and special characters

Based on the number of players, set up each player with a respective color

Start each player's turn

- Check if there are cards in the deck
  - If the deck is empty, reshuffle the deck
- Get a card
- Move on the board based on the card
- Output how far the player is from the castle
- Check if the player has won
  - If the player won, end the game

Classes

Game	• Initialize the game • Go from player to player for their turns • Control the other classes for each turn • Determine if a player has won
Deck	• Make a deck of cards • Reshuffle the deck if the deck is empty • Toss a card if it has been used
Player	• Set a players color • If a player moves on the board, set the players location
Board	• Set up a board with the colors • Calculate based on the chosen card where the player is going to move • Calculate a player's distance from the castle

Code

Main:

```
* File:   main.cpp
 * Author: Heidi Dye
 *
 * Created on October 17, 2019, 6:50 PM
 */

#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include "Game.h"
#include "Players.h"
using namespace std;

/* Notes to self:
 *  To shuffle the cards in the deck: Ouicksort randomized version Algo pg.179
 *      or STL Shuffling Elements pg.589
 *  For the player and their position: sets STL pg.314
 *  For the colors on the board and their numerical representations: maps STL pg. 331
 *  For the Cards: Queues 638
 *  For the Color Position: Stack to subtract the last position with the card position and add that value to the stack STL pg.632
 *  Game Board layout http://datagenetics.com/blog/december12011/index.html
 */
int main(int argc, char** argv)
```

```
{
    cout<<"Welcome to the game of Candyland"<<endl;
    cout<<"How many players? Please enter a value between 2-4"<<endl;
    int players;
    cin>>players;
    //kitty cat proof
    bool valid_play;
    valid_play = players;
    if (!valid_play)
    {
        cin.clear ();
        cin.ignore (10000, '\n');
        cout<< "Sorry, invalid input. Please enter a numerical value\n";
        return 1;
    }
    if(players<2||players>4)
    {
        cout<<"Sorry,value is out of range"<<endl;
        return 1;
    }

    Game game(players);
    //game.pressAnyKey();
    int c;

    fflush( stdout );
    do c = getchar(); while ((c != '\n') && (c != EOF));
```

```
    cout<<endl;
    while(!game.getWinnerPlayer())
    {
        game.playerTurn();
    }

    return 0;
}
```

Game Class:

```
/*
 * File:   Game.h
 * Author: Heidi2
 *
 * Created on October 17, 2019, 8:56 PM
 */

#ifndef GAME_H
#define GAME_H
#include "Board.h"
#include "Deck.h"
#include "Players.h"
#include <string>
#include <stdio.h>
using namespace std;
class Game {
```

```
public:
    Game(int);
    virtual ~Game();
    void initGame(int);
    void playerTurn();    //play true each turn
    void outOfCards(); //reshuffle the cards
    void outputTurn();
    bool winner();
    void removeColor(string, int);
    bool getWinnerPlayer();
    void convertCardColor();
    void pressAnyKey();
    void dos();

private:
    int turn;
    int numberOfPlayers;
    Players* players; //holds the players who are active
    Deck deck;
    string card;
    string cardColor;
    Board board;
    bool winnerPlayer;
    int colorsAvailable;
    string* availableColors;
    string doubleOrSingle;

};
```

```
#endif /* GAME_H */

#include "Game.h"

Game::Game(int numberOfPlayers)
{
   winnerPlayer=false;
   initGame(numberOfPlayers);
}

Game::~Game()
{
   delete[] players;
   delete[] availableColors;
}

//determine which player's turn it is
void Game::playerTurn()
{
   outOfCards();
   //turn++;
   if(turn==numberOfPlayers)
   {
      turn = 0;
   }

   cout<<"Pick a card player "<<players[turn].getPlayer()<<endl;
```

```
    pressAnyKey();
    this->card = deck.getCard();
    //cout<<card<<endl;
    if(board.ifSpecialCard(card))
    {
       cardColor = card;
    }
    else
    {
       convertCardColor();
       dos();
    }
    //cout<<"Card Color: "<<cardColor<<endl;
    players[turn].setLocationColor(cardColor);
    players[turn].setLocationNumber(board.getLocation(cardColor,
players[turn].getLocationNum(), doubleOrSingle));
    //cout<<board.getLocation(cardColor, players[turn].getLocationNum())<<endl;

    if(winner())
    {
       cout<<"Card: "<<card<<endl;
       cout<<"Congratulations Player "<<players[turn].getPlayer()<<" you have won!"<<endl;
       winnerPlayer=true;
       //initGame(numberOfPlayers);
    }

    else
    {
       outputTurn();
```

```
    }
    this->turn++;
}

bool Game::winner()
{
    int location = players[turn].getLocationNum();
    //cout<<"Location "<<location<<endl;
    if(board.getCastleDistance(location)== 0)
    {
        return true;
    }

    else
    {
        return false;
    }
}

void Game::initGame(int numberOfPlayers)
{
    //deck.fillDeck();
    //deck.reshuffle();
    board.setBoard();
    turn = 0;
    winnerPlayer=false;
    this->numberOfPlayers = numberOfPlayers;
    this->colorsAvailable = 4;
```

```
    this->availableColors = new string[colorsAvailable];
    availableColors[0] = "Red";
    availableColors[1] = "Blue";
    availableColors[2] = "Green";
    availableColors[3] = "Yellow";
    players = new Players[numberOfPlayers];
    for(int i=0; i<numberOfPlayers; i++)
    {
        players[i].setActivePlayer(colorsAvailable, availableColors);
        colorsAvailable--;
        //cout<<players[i].getColorTaken()<<endl;
        removeColor(players[i].getColorTaken(), players[i].getPlayerVal());
    }
}

void Game::outOfCards()
{
    deck.ifEmpty();
}

void Game::outputTurn()
{
    //cout<<"Player "<<players[turn].getPlayer()<<"'s turn"<<endl;
    cout<<"Card: "<<card<<endl;
    cout<<"Distance from castle:
"<<board.getCastleDistance(players[turn].getLocationNum())<<endl<<endl;
}

void Game::removeColor(string color, int location)
```

```
{
    string *newArray = new string[colorsAvailable];
    for(int i=0; i<location; i++)
    {
        newArray[i] = availableColors[i];
    }

    for(int i=location; i<colorsAvailable; i++)
    {
        newArray[i] = availableColors[i+1];
    }

    delete[] availableColors;

    this->availableColors = new string[colorsAvailable];

    for(int i=0; i<colorsAvailable; i++)
    {
        availableColors[i] = newArray[i];
    }

    delete[] newArray;
}

bool Game::getWinnerPlayer()
{
    return winnerPlayer;
}
```

```
void Game::convertCardColor()
{
    string colors[6];
    colors[0] = "Red";
    colors[1] = "Purple";
    colors[2] = "Blue";
    colors[3] = "Yellow";
    colors[4] = "Green";
    colors[5] = "Orange";
    size_t found = card.find(colors[0]);
    if (found != string::npos)
    {
        cardColor = colors[0];
    }


    found = card.find(colors[1]);
    if (found != string::npos)
    {
        cardColor = colors[1];
    }


    found = card.find(colors[2]);
    if (found != string::npos)
    {
        cardColor = colors[2];
```

```
    }


    found = card.find(colors[3]);
    if (found != string::npos)
    {
        cardColor = colors[3];
    }


    found = card.find(colors[4]);
    if (found != string::npos)
    {
        cardColor = colors[4];
    }



    found = card.find(colors[5]);
    if (found != string::npos)
    {
        cardColor = colors[5];
    }


}

void Game::pressAnyKey()
```

```
{
  int c;
  printf( "Press ENTER to continue... " );
  fflush( stdout );
  do c = getchar(); while ((c != '\n') && (c != EOF));
}

void Game::dos()
{
    string value[2];
    value[0] = "Double";
    value[1] = "Single";

    size_t found = card.find(value[0]);
    if (found != string::npos)
    {
        doubleOrSingle = value[0];
    }

    found = card.find(value[1]);
    if (found != string::npos)
    {
        doubleOrSingle = value[1];
    }
}
```

Player Class:

```
/*
 * File:   Players.h
 * Author: Heidi Dye
 * Created on October 17, 2019, 8:56 PM
 * Purpose: Generate the players
 */

#ifndef PLAYERS_H
#define PLAYERS_H
#include <stack>
#include <map>
#include <iterator>
#include <algorithm>
#include <string>
#include <iostream>

using namespace std;

class Players {
public:
    Players();
    ~Players();
    //void calcColor();
    void setLocationNumber(int location);
    void setLocationColor(string color);
    void setLocation();
```

```
    int getLocationNum();
    string getLocationCol(); //returns the color of the position
    void setPlayer(string);
    string getPlayer();
    //int getPlayer();
    void setActivePlayer(int, string*);
    string getColorTaken();
    int getPlayerVal();

private:
    string player;    //holds the player color and its respective number
    map<string, int> location;  //holds the player and their location on the board
    stack<string> colorPosition;
    string* availableColors;
    int colorsAvailable;
    string colorTaken;
    int locationNum;
    string locationCol;
    int playerCol;



};

#endif /* PLAYERS_H */

#include "Players.h"

Players::Players()
```

```
{


}


void Players::setActivePlayer(int colorsAvailable, string *availableColors)
{
    this->colorsAvailable = colorsAvailable;
    this->availableColors = new string[colorsAvailable];
    for(int i=0; i<colorsAvailable; i++)
    {
        this->availableColors[i] = availableColors[i];
    }
    int input = 0;
    cout<<"Choose a color:"<<endl;
    for(int i=0; i<colorsAvailable; i++)
    {
        cout<<i+1<<".} "<<availableColors[i]<<endl;
    }
    cin>>input;

    //kitty cat proof
    bool valid_input;
    valid_input = input;
    if (!valid_input)
    {
        cin.clear();
        cin.ignore (10000, '\n');
```

```
        cout<< "Sorry, invalid input. Please enter a numerical value\n";
        cout<<"A default color was assigned"<<endl;
        input = 1;
        //break;
        //input = 1;
    }
    if(input<1||input>4)
    {
        cout<<"Sorry,value is out of range"<<endl;
        cout<<"A default color was assigned"<<endl;
        input = 1;
        //break;

    }
    colorTaken = availableColors[input-1];
    playerCol = input-1;
    setPlayer(colorTaken);
    //return 0;
}

void Players::setPlayer(string player)
{
    this->player = player;
    this->locationCol = "Red";
    this->locationNum = 0;
    location.insert(pair<string, int>(locationCol, locationNum));
}
```

```
void Players::setLocationColor(string location)
{
   this->locationCol = location;
}

void Players::setLocation()
{
   location.insert(pair<string, int>(locationCol, locationNum));
}

void Players::setLocationNumber(int location)
{
   //cout<<"Player Location "<<location<<endl;
   this->locationNum = location;
}

string Players::getColorTaken()
{
   return colorTaken;
}

int Players::getLocationNum()
{
   return locationNum;
}

string Players::getLocationCol()
{
```

```
    return locationCol;
}
string Players::getPlayer()
{
    return player;
}

Players::~Players()
{
    delete[] availableColors;
}
int Players::getPlayerVal()
{
    return playerCol;
}
```

Board Class:

```
/*
 * File:   Board.h
 * Author: Heidi Dye
 * Created on October 17, 2019, 8:04 PM
 * Purpose: generate the board
 */

#ifndef BOARD_H
#define BOARD_H
```

```
#include <map>
#include <iterator>
//#include <algorithm>
#include <string>
#include "Deck.h"

using namespace std;

class Board {
public:
    Board();
    virtual ~Board();
    void setSpecialCards();
    void setBoard();
    void outputBoard();
    int getboardSize();
    int getCastleDistance(int location);
    int getLocation(string card, int prevLocation, string);
    bool ifSpecialCard(string card);
    bool winner(string, int);
    int doub(string, int);
    int sing(string, int);
private:
    int boardSize;  //how many spaces there are on the board
    string board[134];    //holds the varying positions on the board
    int castleDistance; //subtracts the players
    string colors[6];
```

```
    map<string, int>specialCards;
    int location;
};

#endif /* BOARD_H */



#include "Board.h"

Board::Board()
{
    location = 0;
    boardSize = 134;
    //setBoard();
}

Board::~Board()
{

}

void Board::setSpecialCards()
{
    specialCards.insert(pair<string, int>("Plumpy", 8));
    specialCards.insert(pair<string, int>("Mr. Mint",19));
    specialCards.insert(pair<string, int>("Gramma Nut",68));
    specialCards.insert(pair<string, int>("Frostline", 101));
    specialCards.insert(pair<string, int>("Jolly", 41));
```

```
    specialCards.insert(pair<string, int>("Princess Lolly", 91));
}

void Board::setBoard()
{
    setSpecialCards();
    colors[0] = "Red";
    colors[1] = "Purple";
    colors[2] = "Yellow";
    colors[3] = "Blue";
    colors[4] = "Orange";
    colors[5] = "Green";
    int colorCount =0;
    for(int i=0; i<boardSize; i++)
    {
        board[i] = "";
    }
    for(int i=0; i<boardSize; i++)
    {
        map<string, int>::iterator itr;
        for (itr = specialCards.begin(); itr != specialCards.end(); ++itr)
        {
            if(itr->second == i)
            {
                board[i] = itr->first;
            }

        }
```

```
        if(board[i] == "")
        {
            board[i] = colors[colorCount];
            colorCount++;
            if(colorCount == 6)
            {
                colorCount = 0;
            }
        }

    }
}


void Board::outputBoard()
{
    for(int i=0; i<boardSize; i++)
    {
        cout<<board[i]<<endl;
    }
}


int Board::getboardSize()
{
    return boardSize;
}
```

```
int Board::getLocation(string card, int prevLocation, string doubleOrSingle)
{
    //cout<<"Previous Location: "<<prevLocation<<endl;
    int location = 0;
    if(winner(card, prevLocation)&& !ifSpecialCard(card))
    {
        location = 134;
        return location;
    }
    if(ifSpecialCard(card))
    {
        location = this->location;
    }

    else
    {
        if(doubleOrSingle == "Double")
        {
            location = doub(card, prevLocation);
        }
        if(doubleOrSingle == "Single")
        {
            location = sing(card, prevLocation);
        }
    }

    return location;
```

```
}

int Board::getCastleDistance(int location)
{
   return boardSize-location;
}

bool Board::ifSpecialCard(string card)
{

   map<string, int>::iterator itr;
   for (itr = specialCards.begin(); itr != specialCards.end(); ++itr) {
      if(itr->first == card)
      {
         location = itr->second;
         return true;
      }
   }

   return false;
}

bool Board::winner(string card, int prevLocation)
{
   bool win = true;


      for(int i=prevLocation; i<boardSize; i++)
```

```
        {
            //cout<<board[i];
            if(board[i] == card)
            {
                //cout<<"Bye";
                win = false;
                break;
            }
        }


    return win;
}

int Board::doub(string card, int prevLocation)
{
    int location1 = 0;
    int count = 0;
    for(int i=prevLocation+1; i<boardSize; i++)
    {
        if(card == board[i])
        {
            location1 = i;
            break;
        }
    }

    for(int i=location1+1; i<boardSize; i++)
```

```
    {
        if(card == board[i])
        {
            location1 = i;
            break;
        }
    }
    return location1;
}

int Board::sing(string card, int prevLocation)
{
    int location1 = 0;
    for(int i=prevLocation+1; i<boardSize; i++)
        {
            if(card == board[i])
            {
                location1 = i;
                break;
            }
        }
    return location1;
}
```

Deck Class:

```
/*
 * File:   Deck.h
```

```
 * Author: Heidi2
 *
 * Created on October 17, 2019, 10:15 PM
 */

#ifndef DECK_H
#define DECK_H
#include <map>
#include <ctime>
#include <queue>
#include <iostream>
#include <deque>
#include <iterator>
#include <string>
#include <algorithm>
#include "Players.h"
using namespace std;

class Deck {
public:
    Deck();
    void reshuffle();
    void fillDeck();
    void ifEmpty();
    int getDeckSize();
    string getCard();
    void outputMove(string card);
private:
```

```
    int deckSize;   //how many cards in a deck
    //map<string, int>cards;//takes the color of the card are switches it to the numerical value
    deque<string> ogDeck;
    queue<string> shuffledDeck;
};

#endif /* DECK_H */

#include "Deck.h"

Deck::Deck()
{
    fillDeck();
    reshuffle();
}

void Deck::fillDeck()
{
        //single space card
    for(int i=0; i<6; i++)
    {
      ogDeck.push_back("Single Red");
      ogDeck.push_back("Single Orange");
      ogDeck.push_back("Single Yellow");
      ogDeck.push_back("Single Green");
      ogDeck.push_back("Single Blue");
      ogDeck.push_back("Single Purple");
    }
```

```
    //double space card
    for(int i=0; i<4; i++)
    {
      ogDeck.push_back("Double Red");
      ogDeck.push_back("Double Orange");
      ogDeck.push_back("Double Yellow");
      ogDeck.push_back("Double Green");
      ogDeck.push_back("Double Blue");
      ogDeck.push_back("Double Purple");
    }


    //specialty cards
    ogDeck.push_back("Plumpy");
    ogDeck.push_back("Mr. Mint");
    ogDeck.push_back("Gramma Nut");
    ogDeck.push_back("Frostline");
    ogDeck.push_back("Jolly");
    ogDeck.push_back("Princess Lolly");

    //how many cards are in the deck
    this->deckSize = ogDeck.size();
}

void Deck::reshuffle()
{
    srand(static_cast<unsigned int>(time(0)));
```

```
    //shuffle the deck
    random_shuffle(ogDeck.begin(), ogDeck.end());
    //copy the values over into a queue
    for(int i=0; i<deckSize; i++)
    {
        shuffledDeck.push(ogDeck.front());
        ogDeck.pop_front();
    }
    fillDeck();

}

void Deck::ifEmpty()
{
    if(shuffledDeck.size() == 0)
    {
        reshuffle();
    }
}
int Deck::getDeckSize()
{
    return shuffledDeck.size();
}

string Deck::getCard()
{
    string card = shuffledDeck.front();
    shuffledDeck.pop();
```

```
    return card;
}

void Deck::outputMove(string card)
{
    cout<<card<<endl;
}
```

Resources

1. CPP Reference
2. Data Genetics

   http://datagenetics.com/blog/december12011/index.html
3. STL Book for class
4. Algorithm Book for class