Jack Hegman, Gretchen Girdzis, Matt Partridge, Kwaku Nkrumah

Deliverable III: Test Results & Project Status

Status Update (Tasks)

Completed

-Objects have been created to represent game pieces and cards, along with an added deck class.
-GUI has been created on which the game can be played.
-Color classes have not been created, but rather are methods within the computer class.
-Board object has been created.
-Integration of GUI with deck and card class.

On Track

-The opponent (Computer Class) has been created to play the game against, but does not yet handle all of the different cards.
-Have computer class created, still need to create mean computer.
-MySQL reading of data through Java. Waiting on statistics.
-Integration of GUI and computer class.
-Creation of player method in GUI.

Behind

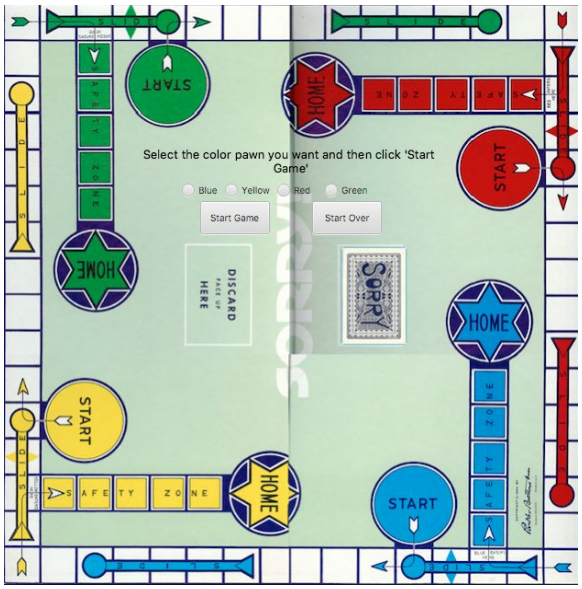
-Overall integration, to implement game flow.
-Having the computer effectively analyze some cards(like a seven).
-Having some trouble figuring how to go about writing the statistics into mysql through Java

Cancelled

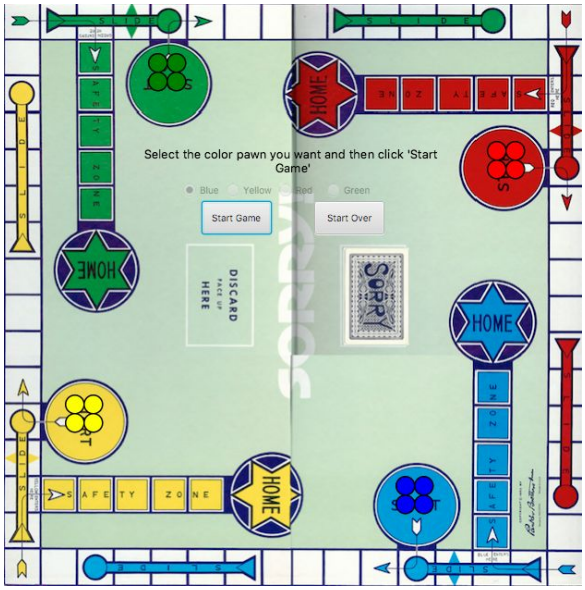
-Player class was an additional task added after submitting Deliverable II, but was cancelled, as the GUI interaction covers the player's actions.

Jack Hegman, Gretchen Girdzis, Matt Partridge, Kwaku Nkrumah

GUI Progress:

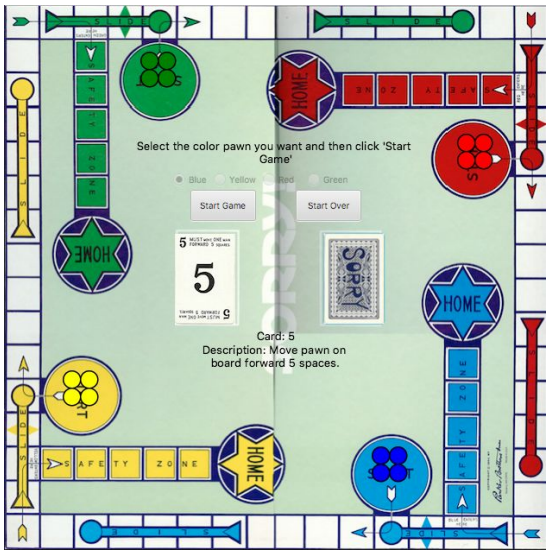


Start of the game. Waiting for user to select the pawn color they want to be.

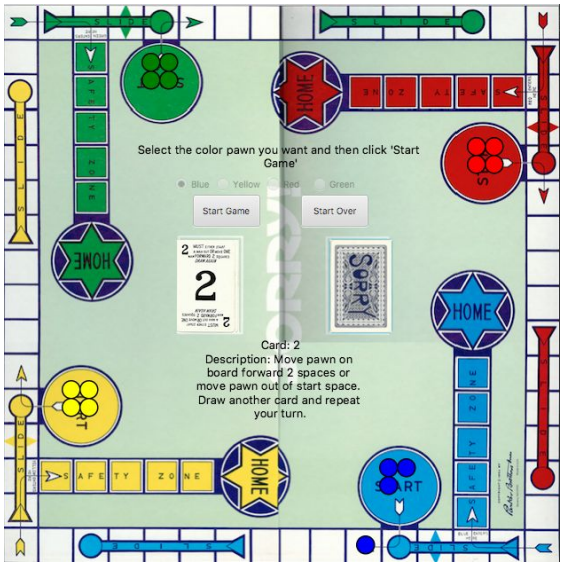


Once user selects pawn color, enable card button and selected pawns

Jack Hegman, Gretchen Girdzis, Matt Partridge, Kwaku Nkrumah



After user selects pawn, spawn all pawns in home spaces and enable the card button.



Click pawn to move.

The grid around the outside is an arraylist of Stack Panes. Pawns move by moving from different Stack Panes. One issue with this is when the pawn gets to the last pane, you get an array index out of bounds error. I fixed this by changing the index back to 0 when the pawn reaches the end of the arraylist.

```
/*
Moves pawn 1 space forward
 */
public void movePawn(Pawn p) {

        if (p.getLocation() == 59) {
            tiles.get(0).getChildren().add(p.getCircle());
            p.setLocation(0);

        } else {
            System.out.println(p.getLocation());
            tiles.get(p.getLocation() + 1).getChildren().add(p.getCircle());
            p.setLocation(p.getLocation() + 1);

        }

}
```

Another issue was moving the pawns from their start home. To fix this, I had to manually set the location for the pawn, and add it to the pane directly outside the start.

```
/*
Moves the pawn from home base out 1 position
 */
public void movePawnFromStart(Pawn p, Color color){
    if(color == Color.BLUE){
        p.setLocation(34);
        disableUsersPawns(bluePawns,p);
        tiles.get(p.getLocation()).getChildren().add(p.getCircle());
        }

    else if(color == Color.YELLOW){
        p.setLocation(49);
        disableUsersPawns(yellowPawns,p);
        tiles.get(p.getLocation()).getChildren().add(p.getCircle());
    }
    else if(color == Color.GREEN){
       p.setLocation(4);
        disableUsersPawns(greenPawns,p);
        tiles.get(p.getLocation()).getChildren().add(p.getCircle());
    }
    else if(color == Color.RED){
        p.setLocation(19);
        disableUsersPawns(redPawns,p);
        tiles.get(p.getLocation()).getChildren().add(p.getCircle());
    }
```

Card Class

Looking at the Card class, each of the cards are stored as static constants. Number and description are stored as variables. Through the Card method, cards are created, through unique numbers.

```
public class Card
{

   //CONSTANTS
   public static final int ONE = 1;
   public static final int TWO = 2;
   public static final int THREE = 3;
   public static final int FOUR = 4;
   public static final int FIVE = 5;
   public static final int SEVEN = 6;
   public static final int EIGHT = 7;
   public static final int TEN = 8;
   public static final int ELEVEN = 9;
   public static final int TWELVE = 10;
   public static final int SORRY = 11;


   //Variables
   private int number;
   private String description;


   //Set card
   public Card(int aNumber)
   {

      number = aNumber;

   }
```

The getNumber() method returns a number associated with a card.

```
   //Get card number
   public int getNumber()
   {

      switch(number)
      {

         case ONE:
         number = 1;
         break;

         case TWO:
         number = 2;
         break;

         case THREE:
         number = 3;
         break;

         case FOUR:
         number = 4;
         break;

         case FIVE:
         number = 5;
         break;

         case SEVEN:
         number = 6;
         break;

         case EIGHT:
         number = 7;
         break;

         case TEN:
         number = 8;
         break;

         case ELEVEN:
         number = 9;
         break;

         case TWELVE:
         number = 10;
         break;

         case SORRY:
         number = 11;
         break;

      )

      return number;

      )
```

The getDescription() method returns the description associated with the card.

```
//Get description on card
public String getDescription()
{

   switch(number)
   {

      case ONE:
      description = "Move pawn on board forward 1 space or move pawn out of start space.";
      break;

      case TWO:
      description = "Move pawn on board forward 2 spaces or move pawn out of start space. Draw another card and repeat your turn.";
      break;

      case THREE:
      description = "Move pawn on board forward 3 spaces.";
      break;

      case FOUR:
      description = "Move pawn on board backward 4 spaces.";
      break;

      case FIVE:
      description = "Move pawn on board forward 5 spaces.";
      break;

      case SEVEN:
      description = "Move pawn on board forward 7 spaces, or move two pawns a total of 7 spaces between them (3 and 4).";
      break;

      case EIGHT:
      description = "Move pawn on board forward 8 spaces.";
      break;

      case TEN:
      description = "Move pawn on board forward 10 spaces, otherwise move pawn backwards 1 space";
      break;

      case ELEVEN:
      description = "Move pawn on board forward 11 spaces, or switch position of any pawn with that of an opponent's pawn, or forfeit.";
      break;

      case TWELVE:
      description = "Move pawn on board forward 12 spaces.";
      break;

      case SORRY:
      description = "Take pawn from own start space, \nplace on space of opponent's pawn, \nand put opponent's pawn in its respective start space.";
      break;

   }
   return description;

}
```

Deck Class

The Deck class extends the Card class, where a Deck object represents a stack of 45 cards in the game. freshDeck() creates a new deck of these cards. getArray() returns the deck. shuffle() shuffles the deck.

```
public class Deck extends Card
{

      final int CARDS_IN_DECK = 45;

      //Create main deck stack and discard stack
      private ArrayList<Card> deck = new ArrayList<Card>();
      private ArrayList<Card> discard = new ArrayList<Card>();


      public Deck()
      {
            super(0);
            freshDeck();
      }

      //Create freshdeck of cards
      public void freshDeck()
      {

         //Offset with 5 1 cards
         deck.add(new Card(1));

         for(int i = 1; i < 11+1; i++)
         {
            for(int j = 1; j < 4+1; j++)
            {

               deck.add(new Card(i));

            }
         }

      }


       //Return deck
       public ArrayList<Card> getArray()
       {

            return deck;


       }


       //Shuffle deck
       public void shuffle()
       {

            Collections.shuffle(deck);

       }
```

The drawCard() method takes the top card and shows it, removes it from the main deck and puts it in the discard pile. Once main deck cards run out, the discard pile recycles.

```
//Draw top card and when run out, recycle cards
public Card drawCard()
{

  Card topCard = deck.get(0);
  discard.add(topCard);
  deck.remove(0);

  while(deck.size() < 1)
  {


     for(int i = 0; i < CARDS_IN_DECK; i++)
     {

           deck.add(discard.get(i));

     }


  }


  return topCard;

}
```

This card and deck class together represent every card in the game *Sorry!*, where cards can be drawn and be implemented by the computer and GUI as to be used by the player.

Board Class:

A print method was added to the board class by printing each individual space so that the position of the slides could be tested. Also this allowed for the position of the computer's pawns to be seen after each move which helped significantly in analyzing how they were moving compared to how they should be moving. The "ATS" stands for "adjacent to space" or in other words, a space from which the corresponding safe zone can be entered.

```
TURN: 1

THE BOARD CURRENTLY LOOKS LIKE:
-----------------------------------------
Space   Type    Occupant            ATS
-----------------------------------------
0       none    blank           none
1       green   slidestart      none
2       green   slide           none    true
3       green   slide           none
4       green   slide           none
5       none    blank           computer
6       none    blank           none
7       none    blank           none
8       none    blank           none
9       green   slidestart      none
10      green   slide           none
11      green   slide           none
12      green   slide           none
13      green   slide           none
14      none    blank           none
15      none    blank           none
16      red     slidestart      none
17      red     slide           none    true
18      red     slide           none
19      red     slide           none
20      none    blank           none
21      none    blank           none
22      none    blank           none
23      none    blank           none
24      red     slidestart      none
25      red     slide           none
26      red     slide           none
27      red     slide           none
28      red     slide           none
29      none    blank           none
30      none    blank           none
31      blue    slidestart      none
32      blue    slide           none    true
33      blue    slide           none
```

Computer Class:

The computer class was tested by adding print statements in under each scenario it went through when determining how to move on a given turn. This way it could be seen whether or not the program was checking the scenarios in the order I wanted. I used the tag "tugwos" and it would show up if the program ran through that section.

```
boolean moved=false;
//System.out.println("tugwos");

//move a pawn out of the start space if able
if((gameBoard.getCompStart()>0) && !(gameBoard.board[4].getPlayer().equals("computer")))
{
    gameBoard.moveCompStart();
    moved=true;
}
//move a pawn into the safe space if able
if((!moved) && (canMoveToSafe(1)!=-1))
{
    //System.out.println("tugwos1");

    gameBoard.moveCompSafe(1);
    moved=true;
}
//move a pawn into the home space if able
if(!moved)
{
    //System.out.println("tugwos2");

    for(int pawn:pawnSafePositions)
    {
        if((pawn+1)==5)
        {
            gameBoard.moveCompHome(pawn);
            moved=true;
            break;
        }
    }
}
//move a pawn onto a slide if it doesn't bump allies
if(!moved)
{
    //System.out.println("tugwos3");
    dontSlide.clear();
    for(int pawn:pawnBoardPositions)
    {
        if(shouldSlide(gameBoard, pawn, 1))
        {
```

For both the computer and the board class I used a sample tester that would have the computer run a specified number of turns(without another player involved) and could either print the previously shown array of all of the spaces, or it could print out just the positions of the computer's pawns after each turn. It also allowed for a pawn to be inserted onto any space so it could be seen how it acted from different situations.

```java
import java.util.ArrayList;


public class tester {
    public static void main(String[]args)
    {
        board b=new board();
        computer c=new computer(true);
        ArrayList userPawns=b.getUserBoardPositions();
        b.printBoard();

        //c.moveNice(b, 1);
        //b.board[2].setOccupied(true, "computer");
        //b.compStart--;
        //b.printBoard();
        for(int i=0;i<5;i++)
        {
            System.out.println("TURN: "+i);
            c.moveNice(b, 1);
            b.printBoard();
            //b.printCompPositions();
        }
    }

}
```