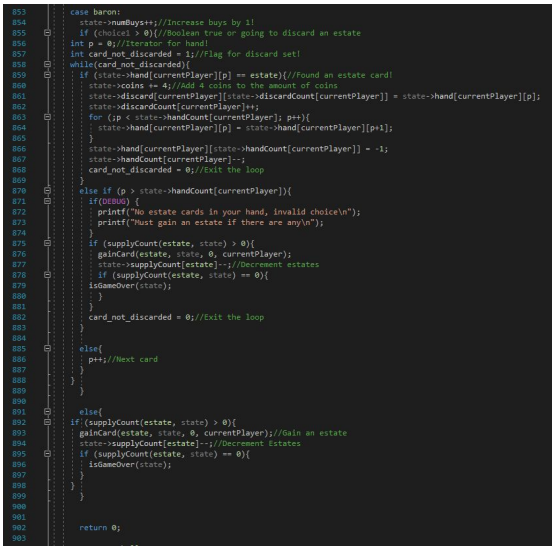
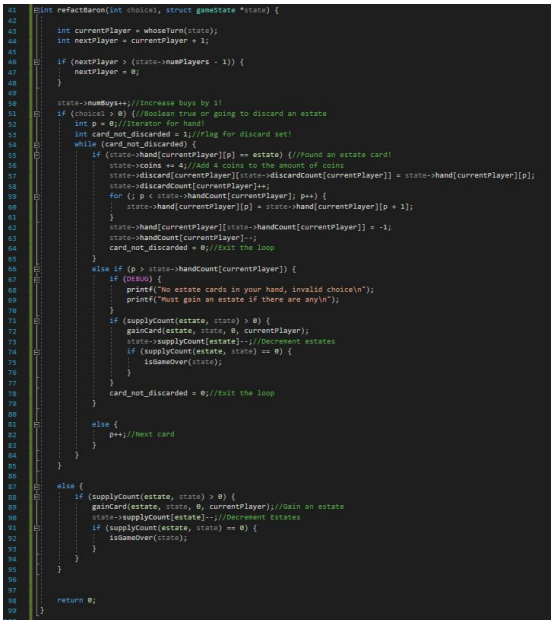
Assignment 2 Kaylene Lim
CS 362 June 7, 2019

Documentation:

a. <u>Refactor</u>

When refactoring the baron, minion, ambassador, tribute, and mine cases in the function *cardEffect*, I created individual functions for each of the corresponding cards. For each card I took the code that existed in the switch case and inputted them in the function. I examined the existing code and made a note of undeclared variables that were declared in the *cardEffect* function before the switch cases. Then, I declared the variables needed. Next, I took a look at the passed variables such as *int choice1, int choice2*, the *struct gameState *state,* and the *int handPos*. If the code from the switch case contained any of the passed variables, the variable is included in the function header and is also passed into the card function. This process is repeated for all the cards and the function declaration replaces the switch case code in the function *cardEffect*. The files were then compiled to ensure that the refactoring has not broken the code.

Below are before → after screenshots of the code

Card	***Before*** *(in cardEffect)*	***After*** *(in independent function)*
Baron		

```
853         case baron:
854           state->numBuys++;//Increase buys by 1!
855           if (choice1 > 0){//Boolean true or going to discard an estate
856         int p = 0;//Iterator for hand!
857         int card_not_discarded = 1;//Flag for discard set!
858         while(card_not_discarded){
859           if (state->hand[currentPlayer][p] == estate){//Found an estate card!
860             state->coins += 4;//Add 4 coins to the amount of coins
861             state->discard[currentPlayer][state->discardCount[currentPlayer]] = state->hand[currentPlayer][p];
862             state->discardCount[currentPlayer]++;
863             for (;p < state->handCount[currentPlayer]; p++){
864               state->hand[currentPlayer][p] = state->hand[currentPlayer][p+1];
865             }
866             state->hand[currentPlayer][state->handCount[currentPlayer]] = -1;
867             state->handCount[currentPlayer]--;
868             card_not_discarded = 0;//Exit the loop
869           }
870           else if (p > state->handCount[currentPlayer]){
871             if(DEBUG) {
872               printf("No estate cards in your hand, invalid choice\n");
873               printf("Must gain an estate if there are any\n");
874             }
875             if (supplyCount(estate, state) > 0){
876               gainCard(estate, state, 0, currentPlayer);
877               state->supplyCount[estate]--;//Decrement estates
878               if (supplyCount(estate, state) == 0){
879             isGameOver(state);
880               }
881             }
882             card_not_discarded = 0;//Exit the loop
883           }
884
885           else{
886             p++;//Next card
887           }
888         }
889           }
890
891           else{
892         if (supplyCount(estate, state) > 0){
893           gainCard(estate, state, 0, currentPlayer);//Gain an estate
894           state->supplyCount[estate]--;//Decrement Estates
895           if (supplyCount(estate, state) == 0){
896             isGameOver(state);
897           }
898         }
899           }
900
901
902           return 0;
903
```

```
41  int refactBaron(int choice1, struct gameState *state) {
42
43      int currentPlayer = whoseTurn(state);
44      int nextPlayer = currentPlayer + 1;
45
46      if (nextPlayer > (state->numPlayers - 1)) {
47          nextPlayer = 0;
48      }
49
50      state->numBuys++;//Increase buys by 1!
51      if (choice1 > 0) {//Boolean true or going to discard an estate
52          int p = 0;//Iterator for hand!
53          int card_not_discarded = 1;//Flag for discard set!
54          while (card_not_discarded) {
55              if (state->hand[currentPlayer][p] == estate) {//Found an estate card!
56                  state->coins += 4;//Add 4 coins to the amount of coins
57                  state->discard[currentPlayer][state->discardCount[currentPlayer]] = state->hand[currentPlayer][p];
58                  state->discardCount[currentPlayer]++;
59                  for (; p < state->handCount[currentPlayer]; p++) {
60                      state->hand[currentPlayer][p] = state->hand[currentPlayer][p + 1];
61                  }
62                  state->hand[currentPlayer][state->handCount[currentPlayer]] = -1;
63                  state->handCount[currentPlayer]--;
64                  card_not_discarded = 0;//Exit the loop
65              }
66              else if (p > state->handCount[currentPlayer]) {
67                  if (DEBUG) {
68                      printf("No estate cards in your hand, invalid choice\n");
69                      printf("Must gain an estate if there are any\n");
70                  }
71                  if (supplyCount(estate, state) > 0) {
72                      gainCard(estate, state, 0, currentPlayer);
73                      state->supplyCount[estate]--;//Decrement estates
74                      if (supplyCount(estate, state) == 0) {
75                          isGameOver(state);
76                      }
77                  }
78                  card_not_discarded = 0;//Exit the loop
79              }
80
81              else {
82                  p++;//Next card
83              }
84          }
85      }
86
87      else {
88          if (supplyCount(estate, state) > 0) {
89              gainCard(estate, state, 0, currentPlayer);//Gain an estate
90              state->supplyCount[estate]--;//Decrement Estates
91              if (supplyCount(estate, state) == 0) {
92                  isGameOver(state);
93              }
94          }
95      }
96
97
98      return 0;
99  }
```

Assignment 2
CS 362

Kaylene Lim
June 7, 2019

Minion

```c
case minion:
  //+1 action
  state->numActions++;

  //discard card from hand
  discardCard(handPos, currentPlayer, state, 0);

  if (choice1)      //+2 coins
  {
    state->coins = state->coins + 2;
  }

  else if (choice2)     //discard hand, redraw 4, other players with 5+ cards discard hand and draw 4
  {
    //discard hand
    while(numHandCards(state) > 0)
    {
      discardCard(handPos, currentPlayer, state, 0);
    }

    //draw 4
    for (i = 0; i < 4; i++)
    {
      drawCard(currentPlayer, state);
    }

    //other players discard hand and redraw if hand size > 4
    for (i = 0; i < state->numPlayers; i++)
    {
      if (i != currentPlayer)
      {
        if ( state->handCount[i] > 4 )
        {
          //discard hand
          while( state->handCount[i] > 0 )
          {
            discardCard(handPos, i, state, 0);
          }

          //draw 4
          for (j = 0; j < 4; j++)
          {
            drawCard(i, state);
          }
        }
      }
    }

  }
  return 0;
```

```c
int refactMinion(int choice1, int choice2, struct gameState *state, int handPos) {

    int i;
    int j;

    int currentPlayer = whoseTurn(state);
    int nextPlayer = currentPlayer + 1;

    if (nextPlayer > (state->numPlayers - 1)) {
        nextPlayer = 0;
    }

    //+1 action
    state->numActions++;

    //discard card from hand
    discardCard(handPos, currentPlayer, state, 0);

    if (choice1)        //+2 coins
    {
        state->coins = state->coins + 2;
    }

    else if (choice2)       //discard hand, redraw 4, other players with 5+ cards discard hand and draw 4
    {
        //discard hand
        while (numHandCards(state) > 0)
        {
            discardCard(handPos, currentPlayer, state, 0);
        }

        //draw 4
        for (i = 0; i < 4; i++)
        {
            drawCard(currentPlayer, state);
        }

        //other players discard hand and redraw if hand size > 4
        for (i = 0; i < state->numPlayers; i++)
        {
            if (i != currentPlayer)
            {
                if (state->handCount[i] > 4)
                {
                    //discard hand
                    while (state->handCount[i] > 0)
                    {
                        discardCard(handPos, i, state, 0);
                    }

                    //draw 4
                    for (j = 0; j < 4; j++)
                    {
                        drawCard(i, state);
                    }
                }
            }
        }

    }
    return 0;
```

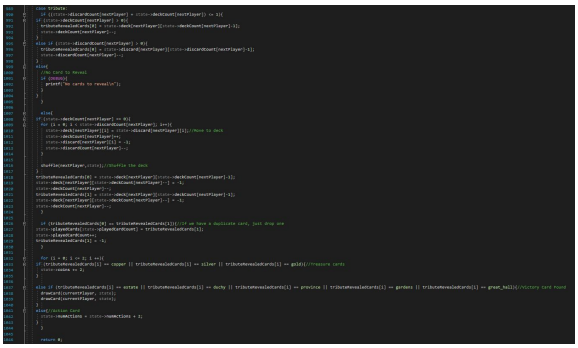
Ambassador

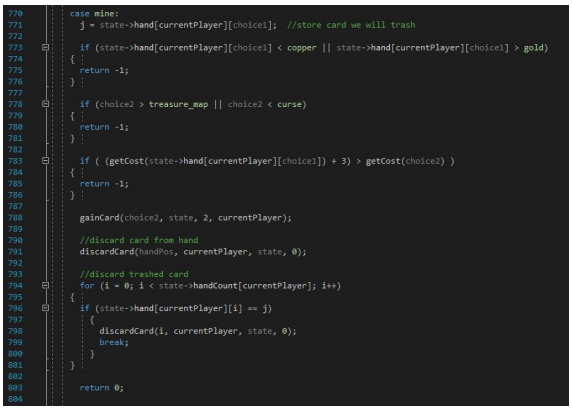
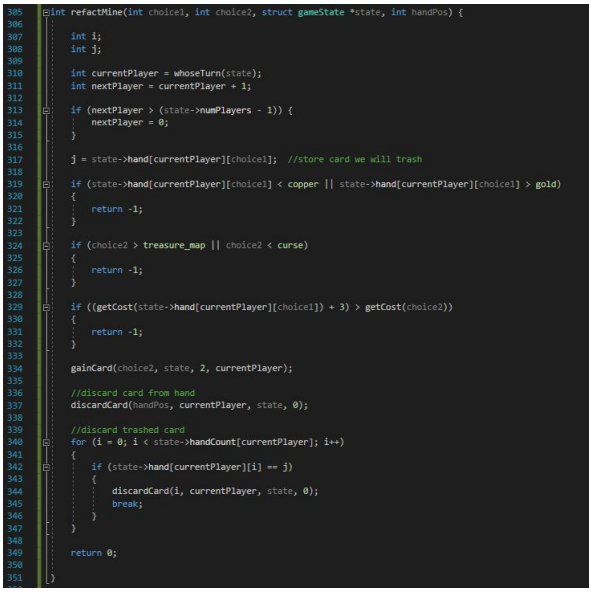
```c
case ambassador:
  j = 0;        //used to check if player has enough cards to discard

  if (choice2 > 2 || choice2 < 0)
  {
    return -1;
  }

  if (choice1 == handPos)
  {
    return -1;
  }

  for (i = 0; i < state->handCount[currentPlayer]; i++)
  {
    if (i != handPos && i == state->hand[currentPlayer][choice1] && i != choice1)
    {
      j++;
    }
  }
  if (j < choice2)
  {
    return -1;
  }

  if (DEBUG)
printf("Player %d reveals card number: %d\n", currentPlayer, state->hand[currentPlayer][choice1]);

  //increase supply count for choosen card by amount being discarded
  state->supplyCount[state->hand[currentPlayer][choice1]] += choice2;

  //each other player gains a copy of revealed card
  for (i = 0; i < state->numPlayers; i++)
  {
    if (i != currentPlayer)
    {
      gainCard(state->hand[currentPlayer][choice1], state, 0, i);
    }
  }

  //discard played card from hand
  discardCard(handPos, currentPlayer, state, 0);

  //trash copies of cards returned to supply
  for (j = 0; j < choice2; j++)
  {
    for (i = 0; i < state->handCount[currentPlayer]; i++)
    {
      if (state->hand[currentPlayer][i] == state->hand[currentPlayer][choice1])
      {
        discardCard(i, currentPlayer, state, 1);
        break;
      }
    }
  }

  return 0;
```

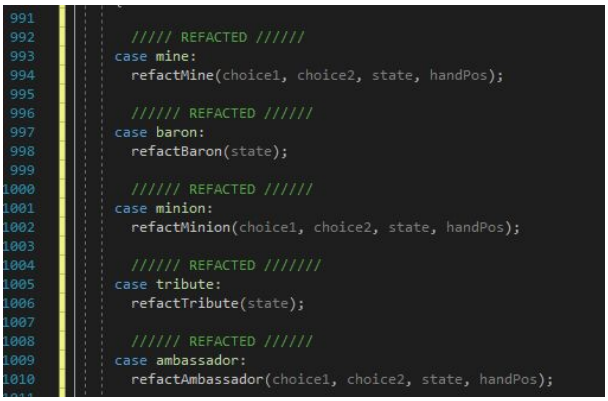
```c
int refactAmbassador(int choice1, int choice2, struct gameState *state, int handPos) {
    int i;
    int j;

    int currentPlayer = whoseTurn(state);
    int nextPlayer = currentPlayer + 1;

    if (nextPlayer > (state->numPlayers - 1)) {
        nextPlayer = 0;
    }

    j = 0;      //used to check if player has enough cards to discard

    if (choice2 > 2 || choice2 < 0)
    {
        return -1;
    }

    if (choice1 == handPos)
    {
        return -1;
    }

    for (i = 0; i < state->handCount[currentPlayer]; i++)
    {
        if (i != handPos && i == state->hand[currentPlayer][choice1] && i != choice1)
        {
            j++;
        }
    }
    if (j < choice2)
    {
        return -1;
    }

    if (DEBUG)
        printf("Player %d reveals card number: %d\n", currentPlayer, state->hand[currentPlayer][choice1]);

    //increase supply count for choosen card by amount being discarded
    state->supplyCount[state->hand[currentPlayer][choice1]] += choice2;

    //each other player gains a copy of revealed card
    for (i = 0; i < state->numPlayers; i++)
    {
        if (i != currentPlayer)
        {
            gainCard(state->hand[currentPlayer][choice1], state, 0, i);
        }
    }

    //discard played card from hand
    discardCard(handPos, currentPlayer, state, 0);

    //trash copies of cards returned to supply
    for (j = 0; j < choice2; j++)
    {
        for (i = 0; i < state->handCount[currentPlayer]; i++)
        {
            if (state->hand[currentPlayer][i] == state->hand[currentPlayer][choice1])
            {
                discardCard(i, currentPlayer, state, 1);
                break;
            }
        }
    }

    return 0;
```

Assignment 2
CS 362

Kaylene Lim
June 7, 2019

Tribute		

Mine	
	`case mine:`
	`j = state->hand[currentPlayer][choice1];  //store card we will trash`
	`if (state->hand[currentPlayer][choice1] < copper || state->hand[currentPlayer][choice1] > gold)`
	`{ return -1; }`
	`if (choice2 > treasure_map || choice2 < curse)`
	`{ return -1; }`
	`if ( (getCost(state->hand[currentPlayer][choice1]) + 3) > getCost(choice2) )`
	`{ return -1; }`
	`gainCard(choice2, state, 2, currentPlayer);`
	`//discard card from hand`
	`discardCard(handPos, currentPlayer, state, 0);`
	`//discard trashed card`
	`for (i = 0; i < state->handCount[currentPlayer]; i++)`
	`{ if (state->hand[currentPlayer][i] == j) { discardCard(i, currentPlayer, state, 0); break; } }`
	`return 0;`
		
		`int refactMine(int choice1, int choice2, struct gameState *state, int handPos) {`
		`int i;`
		`int j;`
		`int currentPlayer = whoseTurn(state);`
		`int nextPlayer = currentPlayer + 1;`
		`if (nextPlayer > (state->numPlayers - 1)) { nextPlayer = 0; }`
		`j = state->hand[currentPlayer][choice1];  //store card we will trash`
		`if (state->hand[currentPlayer][choice1] < copper || state->hand[currentPlayer][choice1] > gold) { return -1; }`
		`if (choice2 > treasure_map || choice2 < curse) { return -1; }`
		`if ((getCost(state->hand[currentPlayer][choice1]) + 3) > getCost(choice2)) { return -1; }`
		`gainCard(choice2, state, 2, currentPlayer);`
		`//discard card from hand`
		`discardCard(handPos, currentPlayer, state, 0);`
		`//discard trashed card`
		`for (i = 0; i < state->handCount[currentPlayer]; i++) { if (state->hand[currentPlayer][i] == j) { discardCard(i, currentPlayer, state, 0); break; } }`
		`return 0;`
		`}`

Switch Cases	N/A	
		`///// REFACTED //////`
		`case mine:`
		`refactMine(choice1, choice2, state, handPos);`
		`////// REFACTED //////`
		`case baron:`
		`refactBaron(state);`
		`////// REFACTED //////`
		`case minion:`
		`refactMinion(choice1, choice2, state, handPos);`
		`////// REFACTED ///////`
		`case tribute:`
		`refactTribute(state);`
		`////// REFACTED //////`
		`case ambassador:`
		`refactAmbassador(choice1, choice2, state, handPos);`

Assignment 2 Kaylene Lim
CS 362 June 7, 2019

b. Bugs

Card	***Bug Description***

Baron	Removed the addition of 4 coins when an estate card is present and discarded in the hand when a baron card is played. The line removed:
	
	state->coins += 4;//Add 4 coins to the amount of coins
	
	The effects of this bug will be apparent when a player has both a baron card and an estate card in hand. When the baron card is played and an estate card is discarded, the amount of treasure will remain the same rather than increase by 4 copper. Otherwise this bug will not be apparent.
	
	Edited code:
	`int card_not_discarded = 1;//Flag for discard set!`
	`while (card_not_discarded) {`
	`  if (state->hand[currentPlayer][p] == estate) {//Found an estate card!`
	`  // add bug here, remove addition of +4 coins when baron card is`
	`  state->discard[currentPlayer][state->discardCount[currentPlayer]] = state->hand[currentPlayer][p];`
	
	I chose to add this bug because it is not very apparent when looking at the code that it is missing. It isn't until the game is played that the bug will take effect. Also, it may be a bug that players don't even notice unless they are very familiar with the game or are very attentive to their deck and money.

	Changed the increment of number of buys to a decrement. The line changed:
	
	state->numBuys--;//Increase buys by 1!
	
	The effects of this bug shows up as soon as the baron card is played as the number of buys the player is allowed is decreased rather than increased! Since this effect is almost immediately seen when playing the baron card, the bug should be realized as soon as the player is trying to buy something and they cannot buy as many as they had expected.
	
	Edited code:
	`state->numBuys--;//Increase buys by 1!, add bug here, decrememnt number of buys rather than increment`
	`if (choice1 > 0) {//Boolean true or going to discard an estate`
	`  int p = 0;//Iterator for hand!`
	
	I chose this bug because it may be easily looked over when glancing at the code but its effects are immediately noticed by the player. This bug may also lead to further faults if buys go into the negative region later on in gameplay.

Minion	Removed the "+1" when defining nextPlayer variable. The line changed: int nextPlayer = currentPlayer + 1; This bug could cause the game to crash as the next player variable is not actually the next

player and the turns would not switch when the minion card is played. When the minion card is played, this bug causes an error in function and prevents any next player data or effects to actually impact the next player. This effect may not be clear until a couple of turns but risks causing the entire game to crash.

Edited code:

```
int currentPlayer = whoseTurn(state);
int nextPlayer = currentPlayer;  // add bug here, nextPlayer does not actually go to next player
```

I chose to add this bug because it is something that may not directly impact the actual effect of the minion card but will affect any data or information that would impact the next player later in the game. This is another bug that is difficult to see without testing but can be caught with a careful eye.

Changed the number from 4 to 5 for other players to discard and redraw. The changed line:

```
if (state->handCount[i] > 4){
        //discard hand
        while (state->handCount[i] > 0)
        {
                discardCard(handPos, i, state, 0);
        }

        //draw 4
        for (j = 0; j < 4; j++)
        {
                drawCard(i, state);
        }
}
```

This bug changes the effect of the card entirely. This bug will come into play when a minion card is played and opponents of the player who have a hand size of 5 cards are not prompted to discard and redraw. Instead of other players having the discard their hand and redraw if they have a hand size of anything greater than 4 cards, they would only do so when they have a hand greater than 5 cards. This bug may not be noticeable with gameplay unless players have a hand size of 5 and are not prompted to discard and redraw.

Edited code:

```
if (state->handCount[i] > 5)  // add bug here, changed 4 to 5
{
    //discard hand
    while (state->handCount[i] > 0)
```

I chose this bug since it is a fairly common mistake which may not impact gameplay right away but definitely changes the card's effect when played. It is a bug that may be caught through static analysis but would be pretty apparent with proper dynamic testing as there is

	a simple error case that can be provided to prove the existence of the bug.

Ambassador	I removed one of the arguments when calling gainCard. The original code is:
	
	gainCard(state->hand[currentPlayer][choice1], state, 0, i);
	
	This bug will go into effect when the ambassador card is played, and the next player has enough cards to discard and when the current player chooses a card for their opponents to gain. The other players will not actually gain that card as there is a mismatch in number of arguments which will cause the game to crash.
	
	Edited code:
	
	`//each other player gains a copy of revealed card`
	`for (i = 0; i < state->numPlayers; i++)`
	`{`
	`  if (i != currentPlayer)`
	`  {`
	`  gainCard(state->hand[currentPlayer][choice1], state, i);  // add bug here, mismatched num of arguments will cause program to crash`
	`  }`
	`}`
	
	I chose this bug because it is a bug that I have regularly found in my own coding projects. Often, this type of bug is found when compiling but sometimes it is not the case and causes a crash in the middle of the code. This is something that can be really hard to see and really hard to tet for when the code continues to crash rather than providing an error or bad output.

	Removed the discardCard line from the if statement. Original code is:
	
	if (state->hand[currentPlayer][i] == state->hand[currentPlayer][choice1])
	{
	discardCard(i, currentPlayer, state, 1);
	break;
	}
	
	This bug would be noticed when the ambassador card is played and the trash copies of the cards are not returned to the supply or discarded from the hand. This bug would be noticed as soon as the players trashed copies remain in their deck which would impact gameplay.
	
	Edited code:
	
	`//trash copies of cards returned to supply`
	`for (j = 0; j < choice2; j++)`
	`{`
	`  for (i = 0; i < state->handCount[currentPlayer]; i++)`
	`  {`
	`  if (state->hand[currentPlayer][i] == state->hand[currentPlayer][choice1])`
	`  {`
	`  // add ug here, remove discardCard(i, currentPlayer, state, 1); making the card not discarded`
	`  break;`
	`  }`
	`  }`
	`}`
	
	I chose this bug because it is difficult to fix something when it is missing. It is hard to see the bug and since it is missing, it is hard to place where it would be appropriate to place it when

	fixing it. This bug also immediately impacts gameplay so it would be very clear when testing that a bug is present and what type of bug but it may be difficult in pinpointing where to fix it.

Tribute	Add an unnecessary integer declaration to the function.
	
	The declared j integer does not do any harm or impact when running the tribute code. This integer is just completely not needed and my clutter the code.
	
	Edited code:
	`int refactTribute(struct gameState *state) {`
	`int i;`
	`int j; // add bug here, variable declaration not needed`
	
	I chose this bug because it is something that may not impact the code right now but may impact it later when additional patches are added to the code. It also effects the cleanliness of the code and shows the coder's attention to detail.

	Changed the shuffle for next player to current player. Original line of code:
	
	shuffle(nextPlayer, state);//Shuffle the deck
	
	This code will come into effect when a tribute card is played. The program would check if the next player had enough cards to display -- the next player is needed to shuffle their deck when they do not have enough cards to reveal. Instead of the next player shuffling, it would be the current player which would be a blatant fault in the game play.
	
	Edited code:
	`shuffle(currentPlayer, state);//Shuffle the deck // add bug here, current player rather than next player`
	`}`
	
	I chose to add this bug because it is very apparent when testing game play of the card's effect. It may also cause the game to crash since the logic that would be needed to lead up to this error would not put the current player in a place where their deck should be shuffled and the next player would not be able to continue playing if they could not shuffle their deck to reveal more cards.

Mine	Changed the curse option to a treasure_map option either way. Original code:
	
	if (choice2 > treasure_map || choice2 < curse)
	{
	return -1;
	}
	
	When playing the mine card, the current player can obtain a treasure map or a curse depending on the value of their *choice2*. Due to this bug, the player will receive a treasure map regardless of the value of *choice2* and would never have the opportunity to receive a

curse.

Edited code:

```
if (choice2 > treasure_map || choice2 < treasure_map)
{
    return -1;
}
```

I chose this since it directly changes the outcome of the game and could also be a very visible bug when looking at the code. Having the code be written as an or statement with the same outcome should be a red flag and show that something is either written poorly or written wrong.

Switched the order of gainCard and discardCard. Original code snippet:

```
gainCard(choice2, state, 2, currentPlayer);

//discard card from hand
discardCard(handPos, currentPlayer, state, 0);
```

This bug may cause an error when the mine card is played and a card is discarded before another card is gained. This can lead to problems in which card gets discarded and how many cards are gained during gameplay

Edited code:

```
//discard card from hand
discardCard(handPos, currentPlayer, state, 0);

gainCard(choice2, state, 2, currentPlayer);
```

I chose to add this bug since the order of codes sometimes doesn't matter and may not affect the code at all. In this case, the effects may be minor but would cause issues if the number of cards gained were affected since the discard happened before.