Isfandyar Rabbani, 932960451
CS 362
Assignment 2

Refactor

I looked for each card in the switch statement and traced the code until it's "return 0;" statement. Between the lines "case <cardName>:" and "return 0;" are considered the function definition. To refactor, I copied the function definition from the switch statement, and pasted it into a global method definition outside of the method the switch statement is called in. Also, I added the variable names that are used into the method parameters so the values could be passed into the function definition.

```
int adventurerCard(struct gameState *state, int drawntreasure, int cardDrawn, int
currentPlayer, int temphand[], int z)
{
	while (drawntreasure < 2) {
		if (state->deckCount[currentPlayer] < 1) {//if the deck is empty we need to
shuffle discard and add to deck
			shuffle(currentPlayer, state);
		}
		drawCard(currentPlayer, state);
		cardDrawn = state->hand[currentPlayer][state->handCount[currentPlayer] -
1];//top card of hand is most recently drawn card.
		if (cardDrawn == copper || cardDrawn == silver || cardDrawn == gold)
			drawntreasure++;
		else {
			temphand[z] = cardDrawn;
			state->handCount[currentPlayer]--; //this should just remove the top
card (the most recently drawn one).
			z++;
		}
	}
	while (z - 1 >= 0) {
		state->discard[currentPlayer][state->discardCount[currentPlayer]++] =
temphand[z - 1]; // discard all cards in play that have been drawn
		z = z - 1;
	}
	return 0;
}


int smithyCard(struct gameState *state, int i, int currentPlayer, int handPos)
{
	for (i = 0; i < 3; i++)
	{
		drawCard(currentPlayer, state);
	}

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

```
int gardensCard()
{
	return -1;
}

int mineCard(struct gameState *state, int i, int j, int currentPlayer, int choice1, int
choice2, int handPos)
{
	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 remodelCard(struct gameState *state, int i, int j, int currentPlayer, int choice1,
int choice2, int handPos)
{
	j = state->hand[currentPlayer][choice1];  //store card we will trash

	if ((getCost(state->hand[currentPlayer][choice1]) + 2) > getCost(choice2))
	{
		return -1;
	}

	gainCard(choice2, state, 0, 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 cardEffect(int card, int choice1, int choice2, int choice3, struct gameState *state,
int handPos, int *bonus)
{
  int i;
  int j;
  int k;
  int x;
  int index;
  int currentPlayer = whoseTurn(state);
  int nextPlayer = currentPlayer + 1;

  int tributeRevealedCards[2] = {-1, -1};
  int temphand[MAX_HAND];// moved above the if statement
  int drawntreasure=0;
  int cardDrawn;
  int z = 0;// this is the counter for the temp hand
  if (nextPlayer > (state->numPlayers - 1)){
    nextPlayer = 0;
  }



  //uses switch to select card and perform actions
  switch( card )
    {
    case adventurer:
                return adventurerCard(state, drawntreasure, cardDrawn, currentPlayer,
temphand, z);

    case council_room:
      //+4 Cards
      for (i = 0; i < 4; i++)
       {
         drawCard(currentPlayer, state);
       }

      //+1 Buy
      state->numBuys++;

      //Each other player draws a card
      for (i = 0; i < state->numPlayers; i++)
```

```
       {
         if ( i != currentPlayer )
           {
             drawCard(i, state);
           }
       }

     //put played card in played card pile
     discardCard(handPos, currentPlayer, state, 0);

     return 0;

   case feast:
     //gain card with cost up to 5
     //Backup hand
     for (i = 0; i <= state->handCount[currentPlayer]; i++){
      temphand[i] = state->hand[currentPlayer][i];//Backup card
      state->hand[currentPlayer][i] = -1;//Set to nothing
     }
     //Backup hand

     //Update Coins for Buy
     updateCoins(currentPlayer, state, 5);
     x = 1;//Condition to loop on
     while( x == 1) {//Buy one card
      if (supplyCount(choice1, state) <= 0){
        if (DEBUG)
          printf("None of that card left, sorry!\n");

        if (DEBUG){
          printf("Cards Left: %d\n", supplyCount(choice1, state));
        }
      }
      else if (state->coins < getCost(choice1)){
        printf("That card is too expensive!\n");

        if (DEBUG){
          printf("Coins: %d < %d\n", state->coins, getCost(choice1));
        }
      }
      else{

        if (DEBUG){
          printf("Deck Count: %d\n", state->handCount[currentPlayer] + state-
>deckCount[currentPlayer] + state->discardCount[currentPlayer]);
        }

        gainCard(choice1, state, 0, currentPlayer);//Gain the card
        x = 0;//No more buying cards

        if (DEBUG){
          printf("Deck Count: %d\n", state->handCount[currentPlayer] + state-
>deckCount[currentPlayer] + state->discardCount[currentPlayer]);
        }

      }
     }
```

```
      //Reset Hand
      for (i = 0; i <= state->handCount[currentPlayer]; i++){
       state->hand[currentPlayer][i] = temphand[i];
       temphand[i] = -1;
      }
      //Reset Hand

      return 0;

    case gardens:
      return gardensCard();

    case mine:
              return mineCard(state, i, j, currentPlayer, choice1, choice2, handPos);

    case remodel:
      return remodelCard(state, i, j, currentPlayer, choice1, choice2, handPos);

    case smithy:
              return smithyCard(state, i, currentPlayer, handPos);

    case village:
      //+1 Card
      drawCard(currentPlayer, state);

      //+2 Actions
      state->numActions = state->numActions + 2;

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

Bugs

Bug 1

The condition of the while loop was altered. Originally the while loop would iterate until the value of drawntreasure was less than 2, implementing the rule of the card that states “Reveal cards from your deck until you reveal 2 Treasure cards. Put those Treasure cards into your hand and discard the other revealed cards.” By changing the while loop condition to 3, the rule is violated and the number of treasure cards received from the turn increases, which gives the user more of an advantage than what is legal.

```
int adventurerCard(struct gameState *state, int drawntreasure, int cardDrawn, int
currentPlayer, int temphand[], int z)
{
       // BUG 1: runs, revealing cards from deck, until drawntreasure is greater than 3.
Correct behavior is: while(drawntreasure < 2)
       while (drawntreasure < 3) {
              if (state->deckCount[currentPlayer] < 1) {//if the deck is empty we need to
shuffle discard and add to deck
                     shuffle(currentPlayer, state);
              }
              drawCard(currentPlayer, state);
```

```
			cardDrawn = state->hand[currentPlayer][state->handCount[currentPlayer] -
1];//top card of hand is most recently drawn card.
			if (cardDrawn == copper || cardDrawn == silver || cardDrawn == gold)
				drawntreasure++;
			else {
				temphand[z] = cardDrawn;
				state->handCount[currentPlayer]--; //this should just remove the top
card (the most recently drawn one).
				z++;
			}
		}
		while (z - 1 >= 0) {
			state->discard[currentPlayer][state->discardCount[currentPlayer]++] =
temphand[z - 1]; // discard all cards in play that have been drawn
			z = z - 1;
		}
		return 0;
}
```

Bug 2

The Smithy card is supposed to draw 3 cards for the user. This bug initializes the value i to 10 in a loop that is only supposed to loop where i is less than 3. Since the value of i is initialized at a value higher than the loop condition, the check will return false and the for loop will not run. The user does not get the number of cards they are supposed to, so this bug would place them at a disadvantage.

```
int smithyCard(struct gameState *state, int i, int currentPlayer, int handPos)
{
		// BUG 2: for loop never runs because i was assigned a value of 10, which means
the first condition check is false.
		for (i = 10; i < 3; i++)
		{
			drawCard(currentPlayer, state);
		}

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

Bug 3

In the Mine card function, the function should exit the program and return a non-successful value (-1) if the value of the current player's hand in index choice1 is outside the acceptable range [4,6] which represents the values between copper and gold. The bug changes the check to the if statement, and only checks to see if the value is outside of the range including copper and silver. Gold could have been available, and if this was the case then the program would exit and the player would not benefit from having gold in their hand.

```
int mineCard(struct gameState *state, int i, int j, int currentPlayer, int choice1, int
choice2, int handPos)
{
		j = state->hand[currentPlayer][choice1];  //store card we will trash
```

```
	// BUG 3: This will run if the value of state->hand[currentPlayer][choice1] is greater than silver (5),
	// which is incorrect gameplay--this would run if the value is gold, which would be in the acceptable range.
	if (state->hand[currentPlayer][choice1] < copper || state->hand[currentPlayer][choice1] > silver)
	{
		return -1;
	}

	// if choice2 is greater than treasure map OR choice2 is less than curse, return -1 (error exit)
	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;
}
```

Bug 4

The discardCard was removed from the function. This means the card is not removed from the current player's deck, and will disrupt how many cards are in the current players hand.

```
int remodelCard(struct gameState *state, int i, int j, int currentPlayer, int choice1, int choice2, int handPos)
{
	j = state->hand[currentPlayer][choice1];  //store card we will trash

	if ((getCost(state->hand[currentPlayer][choice1]) + 2) > getCost(choice2))
	{
		return -1;
	}

	gainCard(choice2, state, 0, currentPlayer);
```

```
	//BUG 4: discard card from hand is missing, not correct gameplay
	//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;
}
```