Class: CS-362
Term: Summer 2017
Author: Jon-Eric Cook
Date: July 9, 2017
Assignment: #2

Refactor

The following cards, that were part of the switch statement in cardEffect, were refactored: adventure, smithy, council_room, village and steward.

Before refactoring began, the inputs and outputs of the selected switch cases had to be identified. This meant, figuring out what variables were required to properly execute the code in each of the selected switch cases. Once the needed variables were noted, the task of creating a function for each selected switch case was quite simple. Simply, all the code in each selected switch case was cut and pasted into its own function.

After a quick review, it was seen that each switch case returned 0. From this, the newly created functions were made to return ints. Upon further inspection, the cardEffect was a function that required an integer to be returned. Due to this constraint, the newly created card functions were changed to return void, rather than an int. The “return 0;” was placed back in the switch case, just below the call to the newly created function.

The Adventure Card function header is as follow:
void adventurerCard(int currentPlayer, int temphand[], int z, struct gameState *state)

As it can be seen, the “adventureCard” function needs “currentPlayer”, “temphand[]”, “z”, and “*state” as inputs. The two variables “cardDrawn” and “drawntreasures” were defined in the “cardEffect“ function. These two variables were searched for and were found to only be used in the “adventure” card switch case. Due to this, they were commented out and copied into the “adventureCard” function.

The switch statement in “cardEffect” for the Adventure Card was updated to be as follows:

```
case adventurer:
        adventurerCard(currentPlayer, temphand, z, state);
        return 0;
```

The Council Room Card function header is as follow:
void council_roomCard(int currentPlayer, int handPos, struct gameState *state)

As it can be seen, the “council_roomCard” function needs “currentPlayer”, “handPos”, and “*state” as inputs. The variable “i” was defined at the top of the “cardEffect“ function. This variable was used for stepping through for loops. Rather than pass it as an argument into the “council_roomCard” function, it was just created as a variable in the “council_roomCard” function itself.

The switch statement in “cardEffect” for the Council Room Card was updated to be as follows:

```
case council_room:
        council_roomCard(currentPlayer, handPos, state);
        return 0;
```

The Smithy Card function header is as follow:

```
void smithyCard(int currentPlayer, int handPos, struct gameState *state)
```

As it can be seen, the “smithyCard” function requires the same inputs as the “council_roomCard” function. It also dealt with needing a variable “i” in the exact same way.

The switch statement in “cardEffect” for the Smithy Card was updated to be as follows:

```
case smithy:
        smithyCard(currentPlayer, handPos, state);
        return 0;
```

The Village Card function header is as follow:

```
void villageCard(int currentPlayer, int handPos, struct gameState *state)
```

As it can be seen, the “villageCard” function requires the same inputs as both the “council_roomCard” and “smithyCard” functions.

The switch statement in “cardEffect” for the Village Card is now as follows:

```
case village:
        villageCard(currentPlayer, handPos, state);
        return 0;
```

The Steward Card function header is as follow:

```
void stewardCard(int currentPlayer, int handPos, int choice1,
                int choice2, int choice3, struct gameState *state)
```

As it can be seen, the “stewardCard” function needs “currentPlayer”, “handPos”, “choice1”, “choice2”. “choice3” and “*state” as inputs.

The switch statement in “cardEffect” for the Steward Card is now as follows:

```
case steward:
        stewardCard(currentPlayer, handPos, choice1, choice2, choice3, state);
        return 0;
```

Bug

Bugs were introduced into the following card functions: adventure, smithy, council room and village.

“Introduce some bug(s) in 4 out of the 5 cards, preferably ‘subtle’ ones that might easily escape a decent test suite.” This instruction, more specifically the key word “subtle”, was interpreted to mean that the introduced bug should be small and cause the card to slightly misbehave. By the bugs being small, it ruled out rewriting the entire functions to create the

bugs. As a result of the 4 buggy cards, the Dominion game as a whole would behave incorrectly.

Determining what could be tweaked in a card's function so that it caused errant behavior was the first step in determining how to introduce a bug into a certain card. This was achieved by stepping through each card's function and analyzing each variable's role and how the flow of execution played out. Once a sound understanding of each function was established, the bugs were then created.

Implementing the bugs was achieved by increasing critical values by 1. These values controlled while and for loops and was also an integer that was being added. These bugs caused the overall game of Dominion to be incorrect because it allowed a player to essentially gain more coins and draw more cards than what was intended.

One way the introduced bugs could be caught would be by comparing an ongoing game with a "perfect" game. The outcome of the "perfect" game would already be known and each step of the ongoing game would be compared to it. Another way to catch the bugs would be to monitor the results of each card played in the ongoing game and compare it to the true actions of that card.

The "adventureCard" function's bug is as follows:

```
// bug -> allows the player to reveal cards from their deck until 3 treasure cards are found,
// instead of only 2
while(drawntreasure < 3) {
        …
}
```

The "smithyCard" function's bug is as follows:

```
// bug -> allows the player to add 4 cards to their hand, instead of only 3
for (i = 0; i < 4; i++) {
        drawCard(currentPlayer, state);
}
```

The "council_roomCard" function's bug is as follows:

```
// bug -> allows the player to add 5 cards to their hand, instead of only 4
for (i = 0; i < 5; i++) {
        drawCard(currentPlayer, state);
}
```

The "villageCard" function's bug is as follows:

```
// bug -> adds 3 to a player's action count, instead of only 2
state->numActions = state->numActions + 3;
```