**Project TP-2.3 - Skunk Project Specific Deliverables**

* Team: John Pham & Rose Xiong
* GitHub:
* We used John’s gitub at “<https://github.com/JohnP665/tp_1_skunk>”.
* Username: JohnP665
* Email: [pham0022@stthomas.edu](mailto:pham0022@stthomas.edu)
* Final TP-2 commit SHA: **86afd4a18ff01812246f3aa0b0bf46952e7c2e74**

**Project Overview:**

For our TP 2 project, we decided to select option one to continue with the Skunk-Specific Deliverables. The Project requirement was to implement multiple players and the Tournament game that allows multiple games. We formed a two-person team: John and Rose. We decided to use Johns Github to submit and work on our main project. We used Java as our main language to code and junit test all our code. The required deliverables that we selected to work on in TP2 was the type of team communications we decided to use, the type of design with GRASP, Two Design Patterns and a Superior separation of presentations logic (UI) and Business Logic (Domain Layer).

**Communications:**

Our team decided to split the work up to a weekly sprint to complete our task. We used zoom for sharing our desktops once a week, phone for quick discussions and outlook to share documents and codes. We also used Saint Thomas’ 365 offices to share our documentations such as our PowerPoints and codes. Our first week we focused on our Domain class. The second week we worked on our Dice and Die class. The third week we worked on the Roll class. The fourth week we worked on our turn class. The five week we worked on implementing the SunkApp and junit test. The six week we worked on refactoring. The 7th week we worked on implementing the single player with one game. The 8th week we focused on the multiplayer game. The 9th week we focused on the TP2 proposal and refactoring. The 10th week we focused on the multi games tournament. The 11th week we focused on the junit test and the final week we focused on refactoring and junit testing.

**Implementation:**

We implemented Skunk project in Java, use Eclipse for IDE, and use JUnit for unit tests.

We implemented the tournament with multiple players. Each round represented each letter of the word Skunk which eventually spelling the word Skunk at the end of the tournament. When a team plays the game the first round would start with the letter “S” and print it out to the console. In the next round “K” would be printed and continues until it eventually spells out SKUNK. This process allowed player to know how many games they have played and how far along they have left in the tournament. Each player is given 50 new chips at the start of every game. If the player has no more chips the player will be removed from the game and the kitty chips will not receive any chips from the player if the player has no chip to give. If there is somehow a tie, then the kitty chips will be distributed evenly. Players cannot roll 10 consecutive times during his or her turn because we do not want a player to roll forever although we have not seen it happened in our game. We also applied a lot of error handling and validation of inputs.

We also create a jar file named SkunkTournament.jar and HowToRun.docx. We use Eclipse project in a remote repository on GitHub and regular commit with Git for our working code.

**Design with Grasp:**

We decided to apply the low coupling by trying to have one domain controller who then sends request to the other classes. We decided to use the tournament as the controller to manage the domain layer. It creates the player and starts the game and manages the game. We also focused on the creator which focused on which class should create the other. We figured that the tournament should know the player and the turns hence it should create the player and the turn objects. The turn then would create the roll and the roll would create the dice and the dice would create the die. I feel this process enabled low coupling by using the controller to create all the players. Once the game starts it calls the turn and the turn will call the rolls and the roll will call the dice. We also refactored our code many times, we removed redundancy to our code. Originally we had the player as the UI which was incorrect because the player apart of the domain layer. We moved the game logic to the tournament because we felt it should be the controller since it should know the player and the turns of each player. We also created a separate classed called the message class to manage all text messages along with the system input and output printstreams. We notice that we called objects that were not being used hence we removed those duplications

**Design Patterns:**

In our Design Pattern we used the Architectural Pattern to help us design our skunk program. It helped us design our highest level of system by us to select the master controller and then selecting all the subsystems to interact with the controller. We also used the prototype pattern to build the message interface class to reduce dependencies without having to create objects for the messages.

**Superior separation of presentations logic (UI) and Business Logic (Domain Layer):**

We created the Messages class as the UI to our Sunk application. We wanted for it to hold all the common text messages and to have one single location for all system incoming coming messages and system out going messages. The output messages are sent to the Messages class and any input is passed from the Message class to the domain layer class. When the game starts the SkunkApp starts the controller which is the tournament. The tournament will then start the Player class and the Turn classes. Messages are then pass to the Message UI to request to start the game and to roll. The active player response is processed by the Message class and passed back down to the domain layer.

**Conclusion**

Overall, we thought the project was a great learning opportunity for us to improve our understanding as well as using better patterns to build our own codes. At the beginning of our class we didn’t know any patterns and we did not down of applying layers or use uses cases for coding. After learning and applying patterns to our project, we felt we have learned a great deal in this software Analysis and design course and improve our coding abilities.