**Status of Project**

**User Interface Subsystem**

* We have implemented MainMenu class; however, we did not start implementing LevelSelect class and OptionMenu class. Thus, user cannot access those panels via MainMenu panel.
* We did not implemented levelSelect class. Therefore, user cannot access GamePanel class with frame. We manually access GamePanel and implement it.
* We add a new class named GamePanel. During implementation we realize that GameEngine class both render, display and run the game and it did not fit our design philosophy. Therefore, we divided it into two pieces. GameEngine class runs the game in 2D array. GamePanel class renders and displays the current game condition. The person who is implementing GameEngine class do not need to know how to display it and the person who is implementing GamePanel class do not need to know how the game works. It helps us to distribute work in our group.
* We start implementing MainFrame class. It initialize the frame components and connects panels. We are working on establishing connections in MainFrame class.
* GamePanel class can render the panel and draw the think we want. Only thing left in this class is paintComponent(Graphics2D g) which will draw the game state according to GameEngine.

**Game Algorithm Subsystem**

* As we explained above, we divided GameEngine class into two pieces. GameEngine class runs the game in 2D array. GamePanel class renders and displays the current game condition.
* GameEngine class can access MapData class and only thing is left for us to create a map according to MapData class and create dynamic objects during run time which is a lot of work; however only this class requires too much work.

**Game Entities Subsystem**

* We have implemented all of the objects in our game. Important parts are done. Currently implementer working taking images by using only one spreadsheet image to organize the implementation.

**Input Management Subsystem**

* We scrapped the idea of making an input management subsystem, instead every class access user inputs via their nested classes.

**File Management Subsystem**

* We split GameData class into two pieces. MapData can only access levels by given level number and ScoreData can only access high scores. We have not implemented yet; however we are planning to create AccessData class to create unlock system for our game. Dividing these classes help encapsulation. If we were using our old system, MainMenu class can access both our map datas and access data.

**New Features**

* In our analysis report, we were planning to add a pause option which original Donkey Kong missing. We successfully added a pause option inside GamePanel class.
* In our analysis report, we were planning to add high score option inside MainMenu which original Donkey Kong missing. We are working on display class when we are done with LevelSelect and OptionMenu classes. Currently MainMenu class can access high score data via ScoreData class. Therefore, we only need to create a user interface for displayment.
* In our analysis report, we were planning to add more levels to the game. Thus, we are coding GameEngine and GamePanel in a way that we will able to add extra levels just by creating a text file named leveln.txt such as level2.txt. We only need to change LevelSelect class to show more levels.
* We decided to not to add a save game option because it does not fit design philosophy of our game.
* We decided to not to add customizations to the game because it does not worth the time to implement.

**Problems**

* During implementation we realize that Game Algorithm Subsystem / Game Visual Subsystem does not fit our design philosophy and it creates a complex class. Therefore we divided them into two pieces.
* During implementing MainMenu class, we realize that making buttons JPanel was not ideal. Therefore we use JButtons.
* During implementation, we realize that some methods of our objects has no purpose, we scrapped them.
* During implementation we realize that every class that extends JPanel uses some sort of input manager. However they are not similar for some classes. Therefore we scrapped whole subsystem and use nested classes for input management.
* During implementation we realize that every class that can access GameData class, both can see our map data and high score data. To solve this problem, we make them separate classes.
* We have not implemented this feature; however we will put small rectangles to our dynamic objects such as Jumpman and enemies. We will decide whether player hit the enemy by using these rectangles. If both rectangles collides, the program will realize that player has hit and enemy. Therefore it will decrease its live points by one.

**Teamwork**

* Object oriented design help working as a team. For example File Management Subsystem and User Interface Subsystem does not depends on each other. Therefore we could able to code some classes from both subsystem at the same time. This is even same for classes in one subsystem. For example we start implementing MainMenu and GamePanel classes at the same time because their job was not depend on each other. They work as their own, but if you combine them they can work as one system as well like a team. Every individuals in a team can work alone, however if you combine them they can work without any dependency to each other.
* Diagrams are useful for teamwork because even though both person think one class should do the same thing, their implementation might differ. Therefore you need a starting point and that is why we use diagrams in computer programming.