Estimate of Person-Hours:

Project 1 Story Approximations:

Meetings: $2 * 1.5 = 3$ (1 hour of meetings about design and plan divided amongst all members)

Stories: $2 * 5 = 10$ (Approximately 2 hours of time spent by each person on their respective components)

Total: 13

Project 2 Story Approximations:

Meetings: 1

Code Reading: $2 + 4 * 1 = 6$ (Member working on AI had to understand more of original code base taking about 2 hours whereas remaining members took less time)

Stories: $2 + 4 * 1.5 = 8$ (Overall, additional functionalities didn't take as much time as Project 1 with exception of AI)

Total: 15

Project 3:

Meetings: 1

System Design and Documentation: 4 (This will be a major portion for this project)

Stories: $2 * 5 = 10$

Total: 15

Design paradigms

We utilized top-down functional decomposition, component-level design, function-oriented design, and object-oriented design.

In the primary brainstorming and design phase, where we do requirements analysis and planning, we considered the project from a top-down functional decomposition perspective. The UML diagram, in the form of a finite state diagram, is a visualization of the primary functionalities and decision flow of the game. This design paradigm is a crucial part of initial design for a prototype system. We must first look at the holistic big picture before anything else. As we are making a game, this is where we look at the end goal that terminates the game and the essential components that determine the flow and repeated actions within the game.

We then break each of the major parts of the game into various components, thus utilizing component-level design. We considered which functionalities would belong in a component with other functionalities and which parts can be separated. For instance, by using basic web languages and technologies, we must separate the user interface from the logic of the project. Generally, we should separate the user interface by having another component that modifies and possibly generates it. Then, we consider a component for controlling the flow and conditions of the game, as it is one of the top-

most aspects of the game itself. Additionally, there are various types of blocks in Tetris, so we must also create an extensible generic component for generating and controlling those blocks. Finally, there should be a grid that will be modified by the game.

By utilizing JavaScript, we are also using largely function-oriented design. We depend on functions that will modify variables and game components to progress the game. The game will move by continuously calling functions. Additionally, when separating parts of the code, we divide and place them into classes and objects, using object-oriented design in organization and improving the readability and creating a coherent code base following basic design principles.

Software Architecture:

Our project most closely follows the 3-Tier software architecture. Each tier will have a set of components that deals with the responsibility associated with them. The different aspects of a component should be designed with enough separation of concerns such that it only performs a role of one given tier at a time. The presentation tier components should deal with user interface, the logic tier should deal with game logic, and the data tier should deal with the game data that will be modified as the game progresses. Firstly, the presentation tier is comprised of the user interface generated through mainly HTML and CSS along with some JavaScript that modifies them based on changes in the game. The logic tier are the functions and member methods of classes that deal with logic decisions during game progression such as the controls, movements, Tetris, and game over. The logic tier functions and methods in turn modifies the data tier. With object-oriented design, the data can be treated as member variables of a class. They do not need to be an entirely separate class or component as operations on the data will be performed by the logic tier. Because of this, the data tier has little reason to be separated from the logic tier, just as the member variables belong in the same object as the member methods which interacts with them. The data tier will be the number representing the speed of the block movement and the grid that holds the data for the empty and filled spaces on the board.

Design Patterns:

JavaScript has prototypal inheritance, so we are restricted from a few creational design patterns. We are likely able to use a builder design pattern for creating various types of blocks and possibly create a prototype for the block. However, creating a block prototype is not a necessity as it is not complex and does not have to contain any additional attributes in a standard version of Tetris; all it needs is to represent some points on the grid. The top-most component will use a mediator design pattern. Its responsibility is to define the interactions between all other objects in the project and their interactions with the user interface. We also may use an observer design pattern. The observer will look for user interface interactions and changes to attributes of certain classes then call other methods or functions and makes changes accordingly. They will be useful for starting the game, earning points, removing blocks during a Tetris, or possibly pausing the game, as an additional functionality.