In this project, we used sandwich integration. The reason for us to use sandwich integration is that we had a bug in the previous project showing up quite late, and we spent a lot of time implementing and debugging it and we want to solve the bug earlier in this project. We could also test the potentially reusable code artifacts. Using sandwich integration can also minimize the weakness of both top-down and bottom-top integration, while able to maximize the strengths of them. In this project, we divided the code into different classes, and have different team members code different classes of the code. Some of the classes are related and must be coded with top-down integration. One example would be the main class and the collision class, which the collision has to be coded after the main class because the main class creates the map of the game, and collision handles the situation when the character runs into walls or gaps. There are several parts that require top-down integration besides main and collision, such as enemy and hurtbox. While some classes are implemented top-down, we also have some classes that we use bottom-up integration. We completed controller class, which is to control the character, before the player class, which is to create the character. The controller has some of the variables required from the player class, but we think we can implement the controller class before the player as we can do it by declaring some global variables. By mixing both integration strategies, we can fix the major design fault earlier, while being able to test the reusable code artifacts at the same time.