**Yuji Sugimoto**

I wrote the code to my game SAT, the 3rd mini-game within the main game. This game in essence features a bunch of SAT books that spawn out of one location and the player must click in order to make them disappear before it hits the bottom of the screen. This used an arraylist for the SAT books so that I could add more and more to the size. Each SAT book would start at the same location, around the middle of the top half of the screen, but since they all were given random velocity and acceleration approached the bottom at different speeds and locations.

A problem I had in order to make the game not impossible was to constrain the speed in which the books were spawning. I did this by setting a variable that kept track of the interval that the books spawned in terms of time. By doing this, I could control the number of books coming in. I could even use this to make the game harder as time went on by making this interval smaller as the game progressed. Another problem was that players could drag the mouse and make the game too easy. To combat this, I created a variable that kept track of time between clicks. I also made a red aiming reticle come up every time the time went down to zero and the player could click again.

I also converted my old raindrops game into a class and made it full screen for the game.

I made the framework for the game menus, win screen, and loss screen. I made sure that all these buttons linked the right game by making every button change a variable “level” that corresponded with each game/ menu. I also compiled some of these games together. Many problems came up in the compiling process that had not been a problem within our own games individually. After linking a few of the games up, I noticed that most of our games did not reset key variables after a loss or win. Because of the way that each mini-game should be replay able by going through the level select, this was imperative that we fix. Also, some games that involved time variables needed some way to reset time because their game was not first. These were a few of the many problems I found while checking for bugs in the compiling process.

Because of the nature of our game, I did not really know how each mini game worked to great detail. For this reason, each member created their own flow chart and kept their own work log for their own respective games. Aside from making sure the project finished on time, I also did some coding and bug checking. This division of work was true for almost all members of the team. We did not really have a fully defined roles because all of us did a little of everything. This worked really well especially because our game involves a lot of other smaller games.

One thing we could definitely do better next time is to work a little faster. Even though we stayed on schedule many of our big problems occurred at the compiling stage. Because we thought that this would be easy, we left only a few days to do it. This problem was compounded by the way we split up the work. We were all worried about our individual mini-games and did not worry too much about the whole. Putting everything together was a pain, and we all thought somebody else was supposed to do certain jobs that we individually should have done ourselves.

**Chris Yang**

I created Whack-A-Grade, a three-round game that tests the reflexes and accuracy of the player. In Whack-A-Grade, three papers pop up onto the screen after a certain period of time has passed. On one of the three papers, there will be a green “A”. The paper is randomly selected each round. The player must click on that paper as quickly as possible. The round does not end until the “A” paper is clicked. The score is dependent upon how much time passes between the beginning and end of the round. After completing all three rounds, the player’s three individual round scores will be averaged together for a final score. If the score is below 450 (meaning 450 milliseconds), the player can move onto the next mini-game of the main game.

My main problem was not a bug after I finished the rough draft of the game, but rather actually creating the rough draft of the game. It took me a long time to figure out how to make the game work with three levels. The trouble was mainly caused by the fact that I had so many mechanisms that depended on time. The flashing of the papers and the recording of the time elapsed for each of the three levels was very confusing for me. In the end, I got it to work by first creating a variable to keep track of the passing time, then three variables to record the start of each level and three variables to record the end of each level.

After finishing the rough draft of my game, however, I did still have bugs to fix. There were numerous ways for the player to cheat. For example, the rounds were started by pressing the spacebar. While testing for bugs, I discovered that even after the round started, the player could press the spacebar again, and the game would think that the player clicked the correct paper. To fix this, I constricted the power of the spacebar so that it could only move from a pause screen to the playing screen and not the other way around.

Also, the level begins with a blank, black screen so that the player does not know where the “A” will be. After one second, the papers flash on the screen and the start time is recorded. Originally, the player could then press the spacebar again to record a new start time. This was a problem because the player could already see where the “A” was. I fixed this by creating a sap (space already pressed) Boolean so that after the player pressed the spacebar once, it had no effect the next times (until the beginning of the next round). These two loopholes were not obvious and probably would not have been discovered by the average player, but I fixed them anyways.

In addition to coding my game, I completed the section of the flow chart that was related to my mini-game. It was a predefined process for running Whack-A-Mole. I also wrote the documentation for my mini-game, tested my game and other games for bugs, and helped group members with their games. Finally, I converted my mini-game into a class so that it could be run smoothly in transition in the main game. I, along with my team members, had so many different roles because of the nature of our game. Since our game was split up into many mini-games which we each worked on individually (for the most part), it only made sense that we should do our own documentation and flow charts. It would be inefficient to have each person learn about the others’ games.

For the most part, this project ran smoothly. Unfortunately, we did not realize how difficult the actual compiling of the games would be. I think we spent a little too much time working on each of our individual games and not enough time planning for the main game. In the end, however, we found a way to make it work, and I think we can learn some valuable prioritizing and time-management lessons from this experience.

**Michael Kretchman**

I created Hallway game. In this game a student is charged with the task of navigating through the hallway of his school without touching any of the walls or the teachers who roam the halls looking for students who are in the corridors. The student, who starts at one corner of the school, uses the mouse to maneuver through the teachers. The teachers move very quickly, so a simple strategy is usually necessary to get passed them. If the student reaches the opposite corner, the player will click the mouse and win the game. This will send the player to the next game. I wrote the code for this game.

The game programming originally consisted of a class called Students, a class called Teachers, and a main code consisting of graphics and calling on functions. The Students classes created the student and defined when it would be eliminated and the Teachers class defined the teachers’ motion. In the end, the game was compiled into three classes so that it could be used in the final game. The hardest part of programming this game was a combination of setting up the teachers’ various motion paths and coding the student’s elimination clauses. Since the teachers motions are in opposite directions, creating their location and velocity PVectors game me some trouble. Since this was slightly confusing, writing the if statement clauses to eliminate the student was difficult because they depended on the location of each teacher. To help solve this problem, I created a simpler version of my code to ensure the concept worked, used a copious amount of comments to ensure clarity, and used a lot of trial and error. In the end, the issue was fully resolved.

Another problem was encountered early on. The game requires the mouse to be in a certain location for the start, so to ensure the mouse would be in the right spot I had to make a start screen for the game with a button that the player would press before beginning the game. This made certain the mouse was in the right spot and that the player wasn’t eliminated before the game even begun.

Besides just coding, I completed a few other jobs. I wrote the documentation for Hallway game, created a portion of a flowchart that organized our team’s entire game, helped other group members with their code, wrote the bug report, and helped check for bugs in the group’s code. Our team was slightly unique in that until the end stage of the project, we were mostly all code monkeys to a large degree. Since we all knew our own codes best, we all completed documentation and predefined processes for our flowcharts that described our own code, and these processes were grouped together at the end of the game. Once the game started to be finished, some members took on other more specific roles. I took on the bug tester role.

As a group, I believe that our team has a good intermediate grasp of processing and its functions. Because of this, we were able to work individually to produce more code in a shorter amount of time. We did this well. The amount of work that was completed in school helped to take a lot of the load off for out of school work. Our team worked well in class. It would have been advantageous to have started more of the bulk of the work slightly sooner, because this would have relieved some stress and allowed a more polished final game to be produced, but nevertheless a suitable final project was completed. The brainstorming process went incredibly well and was very organized. Towards the end our group should have been slightly more organized.