FINAL REPORT

Jeff Hart | Keely Haverstock | Tanner Reid | Sarah Strohkorb
Software Design | Fall 2011 | Olin College

UML Class Diagram

Attached at the end of this document.

A final refinement of the design, including any changes since the previous iteration and an explanation of why the change occurred.

Since our last iteration, we have rearranged the structure of our classes and the code as a whole, added a countdown clock, kept score between the computer and the user, provided level options that control the delay time of the computer, and created a GUI at the end of the game displaying the high scores for that level and the option of playing again or quitting. Below we list all of the changes and why we made them, briefly:

Rearrangement of classes/structure:
Adding functionality to our game was getting harder each time we tried to modify it and there was too much knowledge in some places and not enough in others. We changed the structure to have a central knowledge base (window.py) where the game would be run and control all of the other objects within it (since our game needs a lot of information to be in the same place).

Countdown clock:
We wanted the player to see how much time they have left in a round of the game.

Scoreboard:
We wanted the player to continually be able to see his/her score as well as the computer's score.

Level options that control computer delay:
We wanted the game to work for many different levels allowing for the game to be more user-friendly (players can customize the game to suit their level of competition or skill level).

End of game options GUI:
We wanted to allow the user to A) play again or quit after his/her game, B) see if he/she won or not, and C) see if they got a high score (depending on whether or not they won).

Did you achieve your minimum deliverable? If not, why not? Did you achieve your maximum deliverable? If so, what went right (that is, why was the project easier to complete than you thought)?

We did not meet our minimum deliverable in its entirety, but we did achieve many things beyond our minimum deliverable. The aspects that we wanted to include in our minimum deliverable were A) a user playing against a computer, B) the images on the cards having different rotations and sizes, and C) a score-tracker for the computer vs. user. We did create a game where a user plays against a computer and where the score of the computer and user are tracked. We did not give the ability for the images to be rotated and size-adjusted because we prioritised other tasks over it, especially since making sure that images do not

overlap is a difficult task. The other tasks that we decided to focus on included implementing a game timer to create a definitive end point for the game and creating GUIs for both the start window and the end/high score window.

We did not achieve our maximum deliverable in its entirety, but we did achieve certain elements of our maximum deliverable. The aspects of our maximum deliverable that we did achieve were A) a selection of difficulty levels that would control the delay time of the computer and B) an ending GUI that would display the high scores as well as the option to quit or to play another game. The main aspect of our maximum deliverable that we did not accomplish was networking between computers, which would have enabled two players to play against each other on different computers.

What was the best decision that you made? What would you do differently next time?

We started the project with a poor understanding of class structures which lead to a poor organization of our code. Over the course of this project we made patch after patch in order to make progress and repair mistakes along the way. When we reached the point of the project when we were trying to make the images click-able, the structure of our code made that very cumbersome. We decided to take some time to reorganize our code so that this addition as well as other planned additions could be integrated more easily.

If we were to start this program over, one major change would be planning the structure of the code ahead of time. When we were originally creating the structure of our program we thought of the physical representation of each part of the game. For example, the table, the cards, and the images. We did not think about what was not necessary(table) and what information would have to be stored as attributes of the class to be accessed by other parts of the code. If we were to do this again we would think about the information that needed to be shared and the best way to share it. We may have found it helpful to design the class structure with a ninja since we did not have a good idea of how to organize classes at the beginning of this project.

Were you able to divide the project in a way that allowed the members of the team to work independently and then integrate their work into a whole?

In respect to division of labor, our team employed a combination of individual, paired, and full-team work sessions.

We held regular team meetings during and outside of class, during which we discussed the general direction of our project. This involved outlining goals, planning over-arching object structure of coding, and seeking help for macro- and micro-problems we were encountering.

At our full-team meetings, we generated tasks to be met by the time of our next meeting, and tasks were grouped into two lists and assigned to a pair of teammates. Teammates for each task were determined based on interest in the task and relative skill levels relating to the task involved. Sometimes team members with little skill in a specific area were chosen to gain more skill by working on it, and sometimes team members with great skill in a task were chosen for the sake of time and efficiency, depending the current progress of the outlined goals and planning. We also decided who to place in pairs based on who worked on previous tasks that were closely related to the task at hand. This usually took the form of pairing one person who had previously worked on a linked task with someone who had not, in order to gain experience and understanding of the overall code.

In our paired meetings, partners would set goals for the meeting and work in varied ways to achieve those goals. This usually meant one person typing in a working code file with another person testing certain functions or features of the task at hand. Often, individual work that did not require both partners would be individually-assigned and later combined with the partnered work to form a completed solution for this segment of the code.

If tasks were completed by each pair of partners, we would either implement our solutions by updating code in the repository, or meeting to work out an effective integration of our work. If a task was not completed by a pair, we would meet as a group and try to work out a solution together or seek further help from our mentor or an available NINJA. In either case, a full-team meeting was held to familiarize each other on how we accomplished each task and where to move forward from there.

In general, this allowed for very effective and efficient production and communication of coding solutions, and we are very satisfied with how we handled the division of labor.

What was the most difficult bug you had to find? How did the bug manifest itself? What did you do to find it? How long did it take? What was the cause of the problem? In retrospect, what could you have done to find it faster?

One of the more difficult bugs that we encountered was discovered only recently when the rest of the game was complete. When we completed all of the different components of our game, we began to play it and test it. Everything seemed to be working perfectly. One time, however, the game countdown reached the end, and then began to countdown from the default game time of 1:00 again without bringing up the high score window. Obviously, we quickly realized that this was a problem, so we began to investigate this bug.

This bug turned out to be surprisingly difficult to track because it did not produce any errors and because it did not occur every time we played the game. As we continued testing, we realized that the game timer seemed to skip a second at random points during the game. It seemed that this faulty behavior was correlated to the computer scoring before the human player. This was especially evident when playing the game in “Hard” mode, where the computer scores much more often.

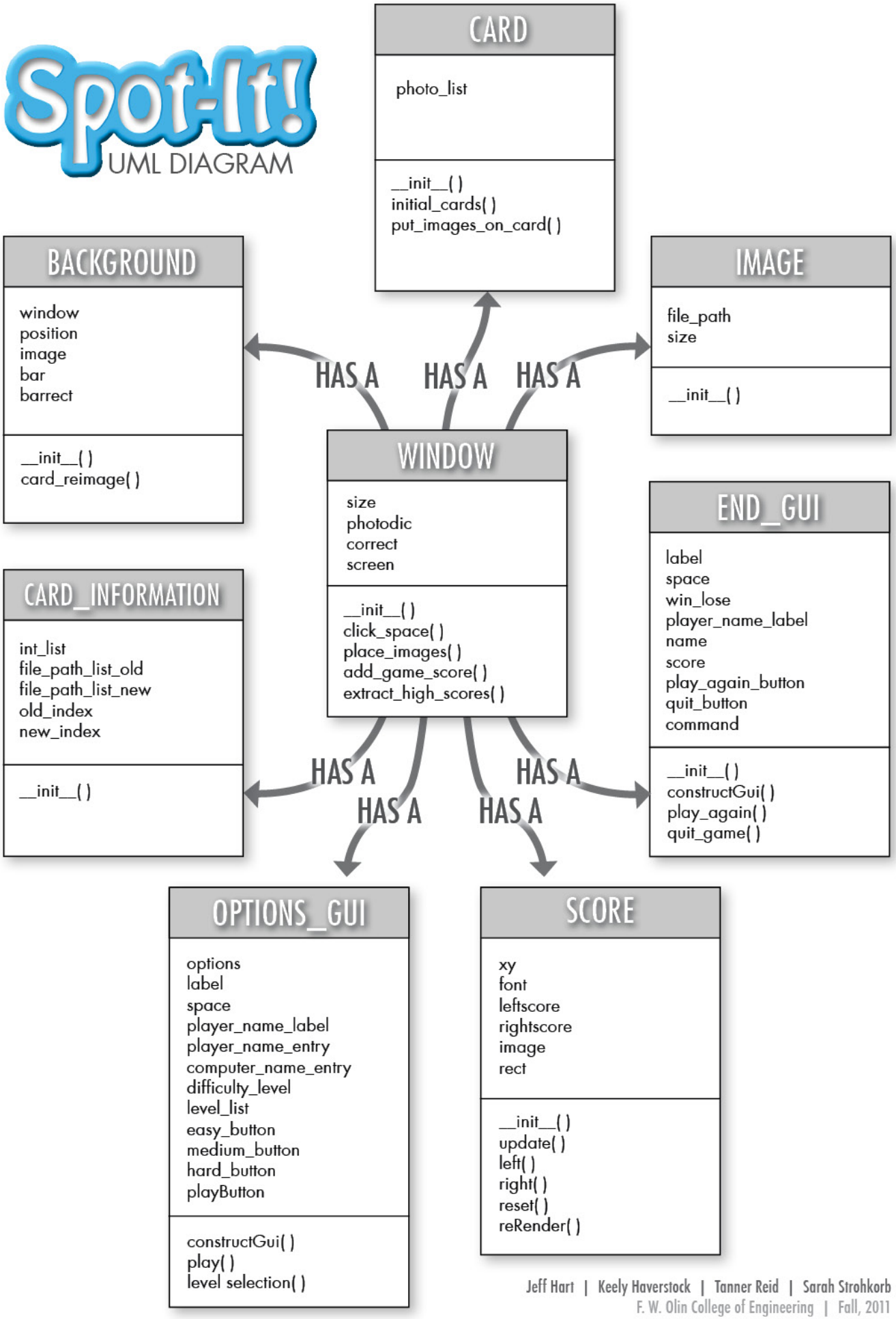
To investigate this issue further, we decided to print the game countdown timer in the terminal. When we did this, we realized that every so often, the countdown timer did, in fact, skip a second. We realized that if this occurred when the countdown should display 0:00 (i.e. if 0:00 was skipped), the countdown would restart at 1:00 without displaying the high score window. This was verified by watching the countdown print in the terminal and seeing it skip 0:00 and begin the countdown again.

We then began to investigate why there might be a correlation between the computer scoring and the countdown timer skipping. We soon realized that the problem could be stemming from the setup of elif statements in the while loop that runs our program. One elif statement handles the COMP_TIMER_EVENT (which is used to trigger the computer scoring), and another handles the GAMETIMEREVENT (which is used to control the countdown timer). It appears that if the COMP_TIMER_EVENT occurs at approximately the same time as the GAMETIMEREVENT, only the elif statement for the COMP_TIMER_EVENT executes. This seemed to be verified by commenting out the elif statement for the COMP_TIMER_EVENT. When we did this, the countdown proceeded as it was supposed to, without skipping any numbers.

However, we have also realized that this issue could be caused by inaccuracy in the pygame.time.set_time() method. This method is not accurate enough to produce the GAMETIMEREVENT in exact 1 second intervals. Thus, it is possible for the GAMETIMEREVENT to skip a second (for example, if one GAMETIMEREVENT occurs at 5.000001 seconds, the next game timer event may occur at 3.999998 seconds). We realize that

this, not the construction of the elif statements, is likely the cause of this bug. As long as we are using pygame.time.set_time(), this error will likely persist. The only way we can think of resolving this issue is by using multithreading and using only the system time to control the countdown clock.

Spot-It!
UML DIAGRAM



Jeff Hart | Keely Haverstock | Tanner Reid | Sarah Strohkorb
F. W. Olin College of Engineering | Fall, 2011