Reindeer Rain

Eric Xu

Course: ICS3U1

Last updated: 2016/01/21

Started: 2016/01/05

Delivered: 2016/01/22

Customer Requirements

The client is Mr. Reid and the requirement is a reindeer-themed game or something interactive that uses graphics. Logic structures, paradigms (procedural and OOP) and lists have to be implemented well. The game needs to have a unique design and work without any errors. The project has to must be well-documented with explanations and charts for high and low level design.

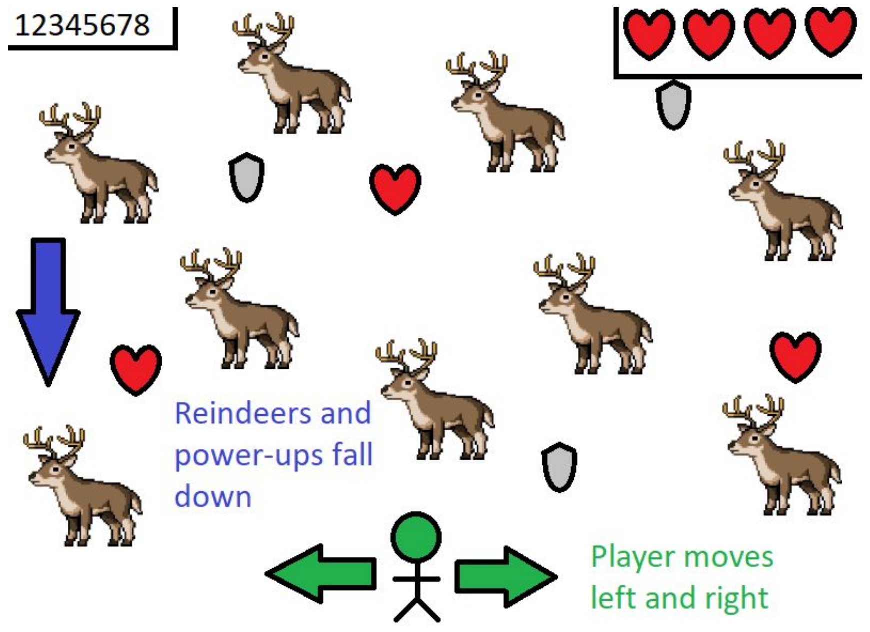
Project Timeline

	Requirements	Design	Implementation	Testing	Deployment
Proposed	Jan 8-9	Jan 10-12, 15	Jan 16-19	Jan 20-21	Jan 22-23
Completed	Jan 8-9	Jan 10-12, 15	Jan 16-20	Jan 21	Jan 22

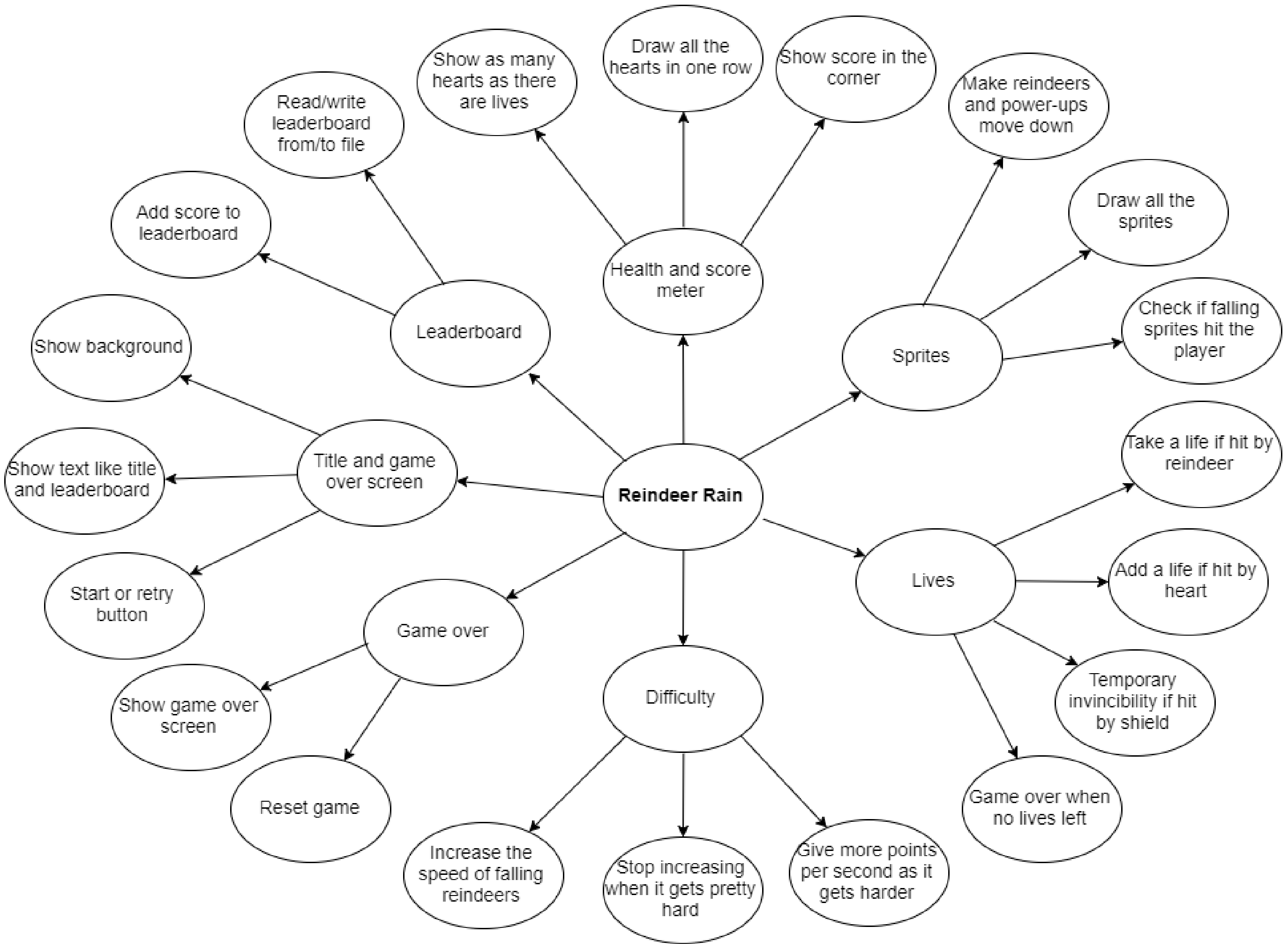
Requirements and design took as much time as expected. However, I underestimated the time it would take to do the implementation. Not much testing was needed because most things worked as expected.

Design proposal

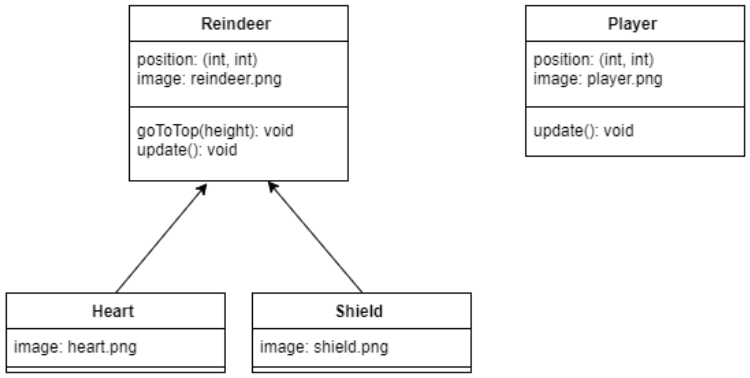
I propose a game called Reindeer Rain where the player dodges reindeers falling from the sky and collects the falling power-ups that restore lives and have effects. The player moves from left to right on the ground. Reindeers and power-ups spawn in random vertical positions and fall from above. The goal of the game is to play for as long as possible before losing all lives.


12345678
Reindeers and power-ups fall down
Player moves left and right

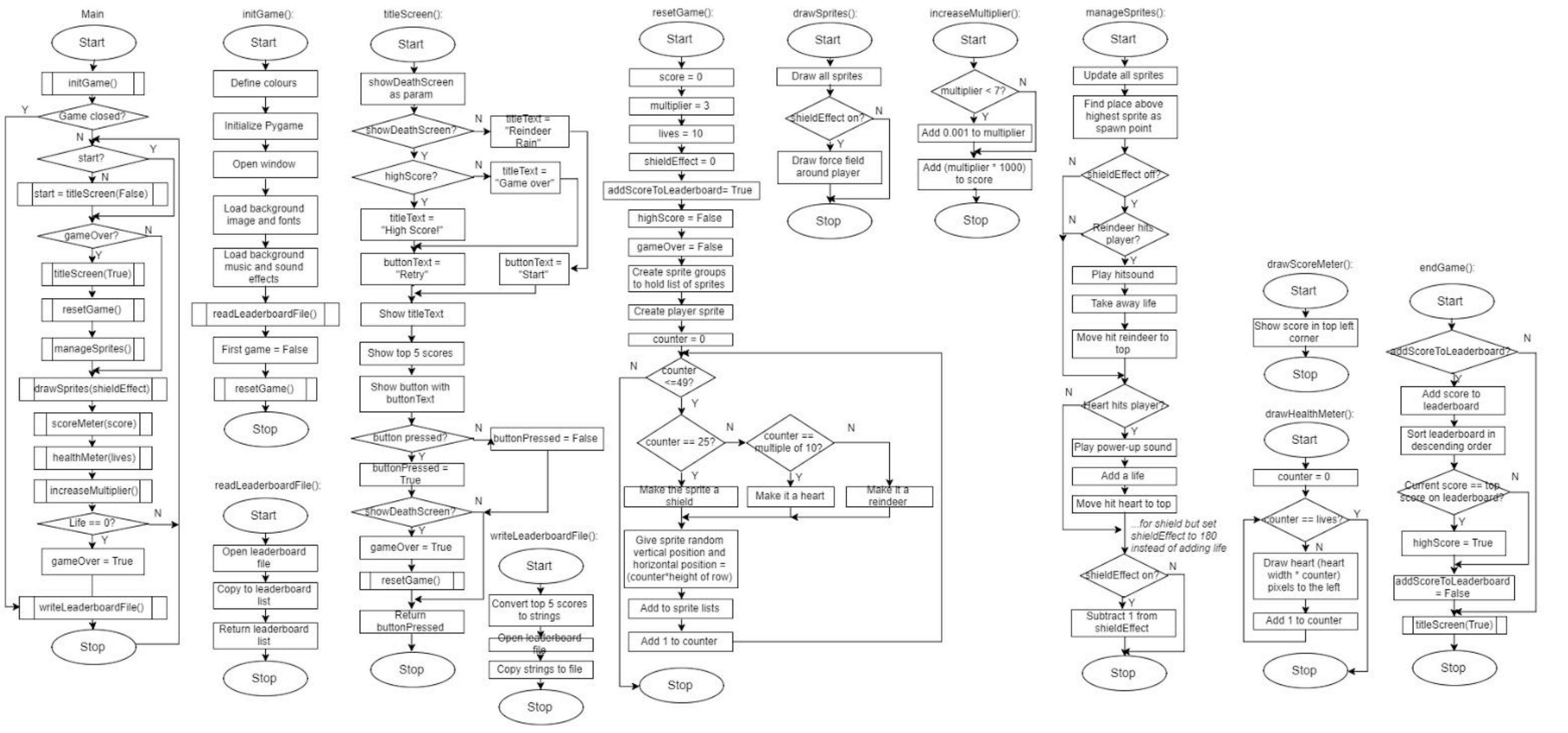
This is a mockup of the proposed game with the major gameplay elements. In the top left and right corners are displays for current score and how many lives are left.


Reindeer Rain
Health and score meter
Show as many hearts as there are lives
Draw all the hearts in one row
Show score in the corner
Sprites
Make reindeers and power-ups move down
Draw all the sprites
Check if falling sprites hit the player
Lives
Take a life if hit by reindeer
Add a life if hit by heart
Temporary invincibility if hit by shield
Game over when no lives left
Difficulty
Increase the speed of falling reindeers
Stop increasing when it gets pretty hard
Give more points per second as it gets harder
Game over
Show game over screen
Reset game
Title and game over screen
Show background
Show text like title and leaderboard
Start or retry button
Leaderboard
Add score to leaderboard
Read/write leaderboard from/to file

This stepwise diagram shows the major procedural components included in the game. They take care of drawing sprites, increasing difficulty, title screens and more. This satisfies the customer’s requirement for a game that implements the procedural paradigm.



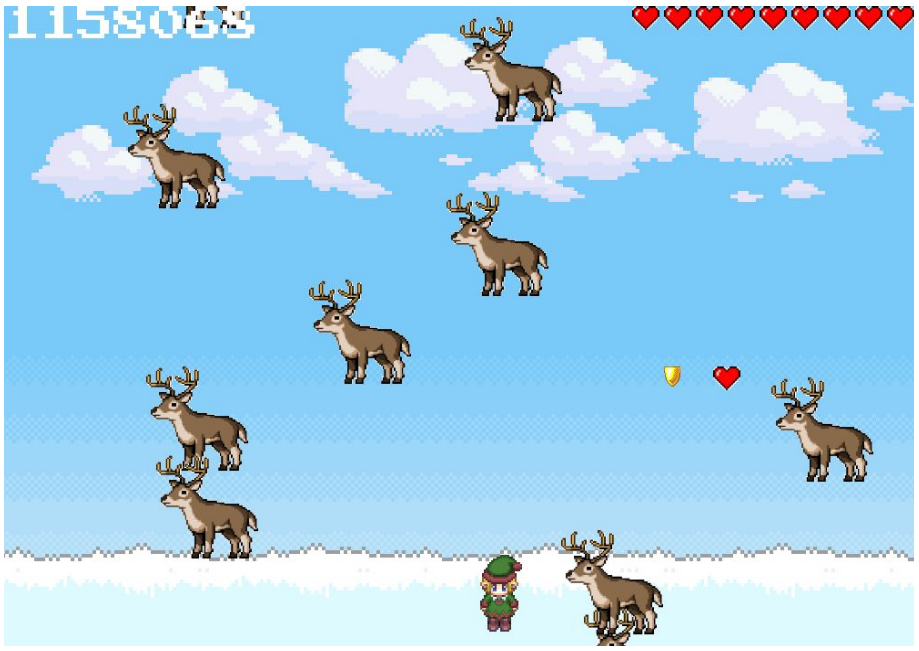
These UML class diagrams represent the classes that will make the sprites in the game. The power-ups (heart and shield) inherit from reindeer because they share two functions that make them fall down and reset position when off screen. The player doesn't share any functions and its update() would move the player sprite left and right instead of down. Solves the requirement for good use of classes and objects.



This is a flowchart outlining the design of the entire game. Variables, selection and repetition are present, fulfilling the requirement for designed use of those elements.

Implementation

Gameplay



The mouse controls the player, moving it left and right. Reindeers, hearts and shields fall from the sky.

A multiplier gradually increases the speed of the falling sprites, making the game harder. The game rewards more points per second as the difficulty increases, but the multiplier eventually hits a limit. The game is over when there are no lives left. The indicator on the right displays how many lives are left and the one on the left shows the current score.

I used what I learned about classes and sprites for collision checking. The game does different things depending on the type of sprite that hits the player, checking separate lists that store different sprites. Reindeers take away lives, hearts add lives and shields add temporary invincibility against reindeers.

Sprites

Sprites are created using classes that inherit from the Pygame sprite class. As shown in the class diagrams, hearts and shields inherit from the Reindeer class because they all share the same functions that make them fall down and spawn. This is how I used what I learned about inheritance to share attributes and functions. However, the player has its own class because it doesn't fall down and only moves left and right. It doesn't have goToTop(), a function that spawns a fallen sprite, and its update() function moves the player differently from the other sprites.

The illusion of reindeers and power-ups falling is created by moving them down until they hit the ground or player, and then spawning them above the screen so they can fall down again. There is always an excess of falling sprites, so they spawn and “pile up” above and off the screen. Falling sprites move down when update() is called each frame. When a reindeer or power-up is offscreen, update() calls goToTop() which spawns it at a height that is above the highest sprite at a random x-coordinate. goToTop() is also called on when a sprite hits the player.

Title screens and leaderboard

When the game is opened, a title screen is displayed that shows the top five scores of previous games. There is also a clickable Start button that turns blue when the cursor hovers above it. titleScreen() also shows the game over screen with different text if the parameter given is True. If the score is on the leaderboard, it is highlighted in yellow. If it’s the highest score, the “Game Over” text is replaced with “High Score!”.

The game stores the leaderboard in a text file to be shown the next time it runs. When the game starts, it copies the contents of the file into a list of ints so it can display it on the title and game over screens. The list is sorted in descending order and new scores are added to it. Before exiting, it wipes the file and copies the top five scores from the list to the file.

Testing

Every time I implemented a feature, I tested it thoroughly to make sure it worked exactly the way it was meant to. For example, when I added the leaderboard feature, I set a modest score and then set a bunch of scores, better or worse than the modest one. This made sure that it sorted the leaderboard, highlighted my current score if it made it into the leaderboard, congratulated me if it was the top score, and displayed only the top 5 scores instead of all of them. This made sure that the program behaved like expected.

One issue that I found after playing the game a lot was that hearts and shields would often spawn underneath or on top of reindeers, and the number of shields was inconsistent. It was because of the way sprites were created before the game starts. There was a loop that ran 50 times, and each time it created a reindeer at a different row each time. However, there was a 1 in 50 chance of also creating a shield but it would be in the same row as a reindeer, so there was a chance they would overlap. Also, because this was based on chance, sometimes there would be no shields or too many. This is an example where fixing problems found by testing made the game better.

Maintenance

Many things went well in this project. I ended up adding more than I expected to the game when implementing the design. I was able to get most things that I wanted to work in a tight schedule. For example, this was my first time that I implemented sprites into a game but I managed to figure it out and make it work well. I was able to make a prototype that had reindeers falling down, but that took too much time to figure out. I also made title and game over screens with clickable buttons.

I spent some time making the graphics look nice. I decided to make the game look and feel 8-bit originally because I could choose from a lot of sprite images that matched in style, but it turned out better than I expected. I also added 8-bit sound effects, a pixel font and I even found a chiptune version of "Rudolph the Red Nosed Reindeer" that matched the theme perfectly.

I am most proud of the leaderboard feature because it contributed more to the game than it might seem. It made it more addictive because people would want to try to beat themselves over and over again as they got better at the game. The game now gives a sense of accomplishment when you achieve a high score. Also, I learned how to read and write files so the leaderboard wouldn't reset each game.

However, there are two things that I would improve if I had the opportunity. I think that there should be more power-ups in the game. The hearts and shields are enough but I would have added a few more power-ups such as a clock that would temporarily slow down the reindeers or a laser that would shoot the reindeers above the player.

Also, I would have added multiple difficulties. I think that the default difficulty is a little too hard for new players, but people that are good at the game would find it too easy and get bored with it. They would be able to choose different levels with different leaderboards.

References

Sprites and background:

http://mercenary-kings.wikia.com/wiki/Deer
http://www.hbgames.org/forums/viewtopic.php?f=10&t=58209
https://yukikootomiye.deviantart.com/art/Sprites-Heart-Life-641296172
http://piq.codeus.net/picture/68684/shield
https://imgur.com/gallery/SELjK

Music: https://www.youtube.com/watch?v=tmAr7DfBy90

Sound effects: https://freesound.org/people/LittleRobotSoundFactory/packs/16681/

Font: http://www.fontspace.com/codeman38/press-start-2p