Snake Revamped

Anna Selonick, Eric Tu, Sarah Miller

Overview of the game

Describe the game itself, inspirations for it, what's new about it. Focus on the experience you intended for the player to have.

For our game, we decided to put our own spin on the classic game "Snake". The player controls a caterpillar/snake character using the arrow keys, and must collect fruits that appear on screen, without hitting any walls or their own tail. The twist is that the fruits all have different effects on the movement of the snake when they appear on the screen.

Our current implementation has 5 "trick power-ups":

- Red slows the caterpillar down
- Blue speeds the caterpillar up
- Rainbow reverses the arrow keys (up is down, left is right, etc.)
- Green starts a timer, giving the user ten seconds to get the green fruit
- Light Blue places three bombs on the screen, which the player must avoid

We intend for this to be a game which is easy to learn, but difficult to master. Since the sequence of trick power-ups is random, no two instances of the game are the same. Players can play over and over before getting bored. Additionally, the trick power-ups take effect as soon as they appear on the screen instead of when the snake eats them. This changes the usual strategy of the game. The player must prepare for what's next when the getting the current fruit. This keeps the player on their feet at all times. Furthermore, the randomly generated trick power-ups move the peak of gameplay earlier in the game. In traditional snake, the first few minutes of gameplay

can be quite mundane and repetitive while the player waits for the snake to get longer so the game can be more challenging. By generating the challenges randomly we hope to make the game exciting from the start. Finally, we hoped that by allowing the player to restart the game by just pressing r and choosing a theme, it would become easy to lose track of time, playing many times in a row. This game is simple and addictive.

Design of the game

What were the key decisions you made in the design? That is, what did you decide to build in order to create software that provides the intended player experience?

In our initial design, we were looking for something eye-catching. We wanted bright fun colors, and cute pictures. To this end, we made a bright green screen, with caterpillars of various colors. When you move along the screen and collect the caterpillars, they append themselves to the head of the snake, so as your snake grows, it also becomes more colorful, and you can see a history of what caterpillars you have picked up along the way. This received positive feedback from users.

One tester suggested that we could have a character selection option, so you could play as something other than a caterpillar. We built off this idea, and created a “block mode” for our game. While the actual gameplay is the same, the aesthetic is very different. The screen is primarily black instead of green, and rather than cutesy caterpillars, the fruits are solid colored blocks. The simpler graphics give the whole game a bit more of a retro feel, but it is still as eye-catching as the original. Players can switch from one mode to the other between games.

Another suggestion we received during playtesting was that our timer power-up should count down rather that up, as it did at the time. We took this into account and now the timer does

just that. Additionally, an improvement we made between the demo and final versions of our game was the inclusion of a high score tracker. This adds to the game by giving the player more of a longer-term goal for replay value. Rather than only collecting fruits for the fun of doing so, the game becomes more competitive by giving the player a concrete number to aim for, whether they are attempting to beat their own record, or a friend's.

Finally we chose to make the trick power-ups take effect when they appeared on the screen rather than when the player eats them to increase player engagement, requiring them to prepare for the unexpected rather than know what's coming. This also why we chose the trick power-ups based on a random number at the time it was generated.

Implementation of the game

What was involved in actually building it? Who did what? Please include source code in your zip archive, and instructions on how to fire it up.

One of the first key decisions we made when designing our game was to use the Panda3D game engine. We decided on Panda3D because it is free, open sourced, uses python, and has an active community should we have wanted to ask questions. As none of us had ever designed any kind of video game before, we wanted to chose a technology that was well documented and had good example code to reference.

Our team met once to twice a week throughout the quarter, and generally did almost all work on the project while we were together in order to receive immediate feedback on ideas, or help if we got stuck. Tasks were fairly evenly split up amongst our group members; we all worked on some of the coding, contributing ideas, working with graphics, etc. We all worked together to build the snake and game world models in Python. We did so based on an existing

open-source example on github called cobra by the user canadien91. We then brainstormed what modifications we wanted to make in our own game. Sarah focused on the interface design and keyboard interactions. She also worked on the timer power-up and the speed power-ups. Eric built in the option to switch themes, the function that randomly generated the fruits/power-ups and worked on the speed power-ups. Anna built the restart function, the bomb power-up, the flip keys power-up and the timer power-up.

Each meeting we generally brain stormed a few new ideas to implement (such as new power-ups, or a new color mode) or improvements that could be made (like having a key of what fruit does what), then spent a few hours making those changes.

What went right

It's important to appreciate what worked well in any process or product. This is where you should take pleasure and pride in what you did well.

At the end of the day, we successfully created a more exciting version of snake. We went from never having made any kind of game (and for some of us, never having used Python at all) to having a fun, functional game to play. We successfully incorporated the feedback we received during the class demo day into the final version of the game, including additional power-ups, different snakes to choose from, and restart functionality. These features enhanced the replay value of the game.

We received a lot of positive feedback during the final project party and the initial class presentations. Players liked the added challenges, especially the trick power-up that flipped the keys. As planned, they had to change their usual snake playing strategies once they noticed that the power-ups took effect when they appeared on the screen. For example, staying away from the

walls in the event of power-up that flip keys happens. The trick power-ups seemed to change the game enough that players could play for ten minutes and still feel challenged and engaged.

Finally, in terms of constructing the game, we allotted a significant amount of time to building and understanding the snake model as a team. This made it a lot easier to add in the trick power-ups. The modular structure of the code made it a lot easier to alter different settings of the game for specific power-ups, like the speed of the snake, without affecting the entire codebase.

What went wrong

One of the key lessons computer science teaches us is that bugs happen. They can happen at the highest levels of the design to the lowest bits of the implementation. By reflecting, we can find them and fix them (or at least figure out how to do better next time). So, what went wrong in the design and/or development of your game?

As can be the case in most fields, we found during this project that some thing that sound easy in theory are actually pretty hard to implement. For example, we figured it would be easy enough to just add a "Press 'R' to restart" type of function, but in practice this was not such an easy task (although we were able to implement it). Building the restart function was a good test of how well designed our code was because each node we had created needed to then be deleted, so we would have to keep track of all of our objects efficiently. The code was designed well enough that we were able to implement a restart function. However we probably could have benefitted from creating a trick-power-up class with subclasses of each type of power-up rather than having so many global variables attached to our world instance that indicated whether a trick-power-up had been triggered. These global variables were probably the reason behind one

of the bugs that we didn't catch until the final round of playtesting during the game party in which the bombs didn't show up if they were triggered after other power-ups.

Something we attempted, but ultimately did not succeed in implementing was multiplayer mode. We wanted to have 2 snakes on the screen at once, with one being controlled by the arrow keys and the other by WASD. The two snakes would be forced to race each other to the fruit, and the one who got the most would win. After debugging many strange issues, we decided to abandon this idea. In the end, we decided that we could better utilize our time making other significant improvements, as we weren't even sure how much this would actually add to the user experience.

Ideas for next time

What would you do differently next time you design and build a game? Think in terms of both debugging what went wrong, but also in terms of what the next steps might be, if you were to do another iteration of your game.

If we were to continue to develop this game, one good direction to head in would be multiplayer functionality. We started to work on this idea, but did not have time to fully realize it. It would be interesting to have a mode in which two players sit side by side and play head to head using one computer. One could control the arrow keys, and the other could use WASD. This would open up new strategies, in either how to most quickly reach fruits, or how to block off one's opponent so they ran into something while you could get the fruit. Additionally, we could explore the idea of having the negative power-ups affect the other player instead of the one who obtains it. It could also be fun to create a multiplayer mode which did not require the players to be in the same room, but rather let them challenge each other over the internet.

What we would do that would be different if we were to design and build a new game is when starting off to choose a game engine that has even more high-level functions and tools. We were a bit limited in designing our game due to restrictions with coding in python. But if we used a higher level game engine, we could spend more time manipulating objects that are already created, dragging and dropping obstacles (like some other groups have), and play around with some more original ideas.

We would also outline the details and semantics of the game in the beginning: what experience do we want to player to have, what functions is going to achieve the goals and make the game interesting, etc. We started off with very general ideas and toyed around with things that could be done within our scope but played it by ear. In the future, we would like to be more

very clear with what we are working towards in the beginning. We have a better sense of what to ask the players during feedback and ideas of things that they look for.