Sketch-a-Level

M. Yvonne Hidle
Carnegie Mellon University
827 Collins Ave
Pittsburgh, PA 15206 USA
yvonnehidle@gmail.com

Abstract
Sketch-a-Level is an android application that allows users to draw their own maze for a game inspired by the seminal Pacman. This paper is a short write up on the entire process from the initial concept to the end product.

Introduction
Sketch-a-Level was an exploration into the idea that users should be able to generate or "sketch" their own game levels. Initially the project was a literal interpretation of this concept with a more physical user interaction and more emphasis on hardware and computer vision. Later, as hardware limitations and issues related to the computer analyzing complex analog data were encountered, the project transitioned into a completely screen based interaction in the form of a simple and fairly straightforward Android application for touchscreen devices, primarily tablets.

This paper is mainly about concept and process, detailing the entire project from start to finish. It highlights the problems, attempts made to solve said problems, and how this impacted the evolution of the project over time.

Process: Original Idea
The project started out with the idea that an ordinary piece of paper could be "activated" or augmented by being placed on a special surface or rig.

One attempt to accomplish this consists of a rig with a standard glass table, a Kinect, a projector, a computer with openFrameworks, a piece of paper, and a black marker.

Game Control
The main hurdle with this setup was related to game control. The original idea was that a user could control their character using their finger. While it would have been much easier to use game controllers the inherent problem with game controllers is that they are too detached from the natural movements of the human hand and finger. Game controllers also introduce

another object that acts as an intermediary preventing the user from interacting directly with the maze they have created on the piece of paper. The most direct way for the interaction to occur without physical interference is to track the motion of the hand and finger with a video camera as it travels along the user generated maze. This method involves the use of openFrameworks, the ofxKinect addon, and a Kinect camera fitted with a pair of +2.5 reading glasses due to the close proximity of the play surface to the camera.

The first attempt, met with moderate success, is to modify the ofxKinect blob detection code to suit the purposes of detecting finger motion. The first step is to isolate the tip of the finger and get it to track the finger with relative accuracy. However, there are a few problems with this method. While the Kinect fitted with +2.5 reading glasses is able to see closer than normal, it still cannot see in the 0-2 foot range. This is a problem since the camera needs to be closer to the paper in order to generate a collision map image with a resolution that is high enough to be usable. Another issue with the Kinect is that the camera's inability to see closer than 2 feet makes it difficult to isolate the finger consistently. This leads to long periods of time when the finger completely disappears depending on the proximity of the user's hand relative to the camera. These issues led to the second attempt at game control using a standard webcam instead of a Kinect.

Three approaches were taken with the webcam:

1) Blob detection paired with contour tracing and background subtraction.

2) A finger tracking add-on in Processing.

3) Basic color detection

All three methods had their fair share of issues, most of which revolved around speed and accuracy. The finger tracking add-on was very slow, even after the code was optimized by limiting the number of points and simplifying the contour to the detriment of accuracy. Blob detection and contour tracing with background subtraction works fine in openFrameworks, but could not be used for the gameplay because the game program is written in Processing. As such the method of finger tracking needs to be compatible with processing so the game code does not have to be re-written. The third approach was color tracking. Color tracking was used due its simplicity and its compatibility with the game code and Processing. Color tracking was the last game control option pursued before switching the whole project onto Android.

Collision Mapping

The second major component of the game interaction involves detecting the maze sketch, saving the maze sketch, and converting the maze into a collision map that digital characters can interact with. The ability to use a finger or pen for maze creation is paramount given that the basis of the game is a user generated sketched maze. If the user has to use a mouse or a game controller to sketch the maze then the process of sketching becomes more cumbersome and less spontaneous as it gets more difficult to create mazes.

Initially the idea was to have physical buttons on the game table that would be labeled “save map” and “start game.” The user gets a piece of paper, sets it on the table, draws a maze and hits the save map button. Once the aforementioned button is pressed the computer takes a snapshot of the maze (preferably without the user's hand in it), analyzes the image, and then loads the analyzed image into the game as a map file. This method works fine until it is paired with the projection used for gameplay. When the projector is on and running it is impossible to get a good clear photo of the drawn maze. Two resolutions are effective at ameliorating the impact of the projection on the image capture quality:

1) Add additional lighting to counterbalance the projection. This worked well but made the rig feel bulky and cumbersome as the lighting had to be in the same space that a user would normally position their hands relative to the paper.

2) Have an apparatus that would block the projection when drawing and saving the image, and then unblock the projection when the game was started.

Projection
Due to limited availability of equipment, an old Optima projector was used for the majority of experiments. The projector has the misfortune of very poor image resolution and an extremely long throw. For this project a projector with a short throw and good image clarity is very important. The projector should be able to be used within a rig such as a Reactable table.

However, in the attempt to work with a readily available projector, the best resolution is to use a mirror in order to enable the placement of the projector about four feet from the bottom of the glass surface. This distance is approximately what is needed in order to project the gameplay image onto an 8.5x11 sheet of paper on the glass table. Unfortunately, due to the focal length of the projector, the image quality is grainy and details are for the most part illegible. In addition, the throw is far too long and the projector cannot be placed under the table, making a table-sized rig impossible.

Issues overall
The original idea was abandoned for three main reasons:

1) The rig would need to be excessively large in order to accommodate the throw of the projector and additional embedded lights. This would be fine if the entire table surface was intended to be used for interaction. However, the intention was to make the standard and commonly available 8.5"x11" piece of paper interactive.

2) The quality of the projection was very poor. The characters were extremely illegible.

3) The game control was poor. A finger needs to be able to be tracked accurately as long as it is within view. When using the RGB feed the camera had difficulty tracking a finger due to the light noise of the projector. When using Kinect the distance between the camera and the table is too great.

Process: Re-boot
After struggling to resolve all of the aforementioned issues simultaneously, it was clear that an alternative means of finger tracking was necessary. I had done

work with capacitive sensors in the past and was inclined to use them again. However, it was brought to my attention that I was essentially trying to recreate a tablet… With that in mind I decided it was in my best interest to switch the project to be entirely screen based on the Android tablet that I already have. I felt the transition was not ideal but still preserved the essence of sketching a maze quickly and easily.

Process: Android application

The final product ended up being an Android application that consisted of two primary phases:

1) A drawing application that allowed the user to draw maze walls, create passageways, portals, traps, and other special features.

2) The actual game where users could control a character and see how the AI responded to their maze.

With everything related to computer vision aside, the new obstacles are all related to rewriting all of the game code to work on Android. The most difficult struggles are with ghost AI and game optimization. Given the limited memory of an older tablet, “out of memory” errors related to the number of images were a persistent problem. While this error is mostly resolved, there are still quite a few issues with speed and overall frames per second.

The processing speed limitations of the tablet are an issue because of the need to blur the drawn mazes so that the ghosts and cat could interact with the image appropriately. During the exhibition most users were not patient enough to wait for the image to “format” or blur. Because of the lack of a loading bar (or other indicator), most people felt the game had frozen, when in reality it was still blurring the image.

Another major issue is character control. The most straightforward approach is to have the character follow the position of the user’s finger using a simple easing function and brightness detection to prevent the character from going through walls. The two processes are very effective but have several issues while they are run simultaneously. For example, the character moves flawlessly when it ignores the walls (the brightness detection is off). But once wall detection is introduced, the user loses control or has difficulty moving the character when the character encounters a wall.

The frame rate is another major issue that prevents the use of a lot of intense but more effective functions. The frame rate needs to be around 30 frames per second in order to keep the game from being choppy, but this is difficult when using an image as a background and having multiple characters constantly checking the pixels around them to decide where to move.

Overall I believe the project turned out fairly well given the time constraints, issues faced early on, and a major user interface change near the end of the project.