**[I’ve written this as a more of a reflective summary, I wasn’t sure if was meant to be a technical report]**

Kinect Input

The Kinect SDK allows us to register for different types of data to get from the device. These are: colour stream, depth stream and skeleton data. We only get the skeleton data since this gives us hand location data. However, this requires the full skeleton to be tracked to just get the hands. This means we can’t track a single hand on its own. We could use the depth input and try to find hand locations ourselves, but this is extremely hard for little gain, and it probably wouldn’t be as accurate as the skeleton data.

We added the ability to adjust the scaling of the Kinect input into an option in the configuration file. This allowed for adjustment for people of smaller height. We chose to make this a configuration option rather than a hardcoded value because the placement of the Kinect in relation to the user can affect the user’s ability to reach parts of the screen. We also scale the x and y axis input separately since people can usually reach from left to right easier than reaching vertically.

One disadvantage to this scaling is it adversely affects the accuracy of the Kinect. The Kinect sensor does not track the user perfectly, especially the extremities such as the hands. As such the position of the person’s hand is constantly in flux, even when the real hand is stationary. Increasing the scaling factor of the Kinect amplifies this inaccuracy. This can cause the hand to excessively jitter, and makes clapping much more difficult. As such, the values should be left to as low as is still usable.

We did find that the angle of the Kinect and distance from the Kinect affected the difference in height greatly. Having a direct angle at mid-distance provided the most consistent usability between users.

There are some hardware limitations of the Kinect however. For example, users in wheelchairs may have some difficulty in using the Kinect. Because we use skeleton tracking to find hand positions, if the Kinect has issues tracking the full user, then the hands can’t be displayed. The Kinect will often interpret a wheelchair or crutches as legs, but this may not be a significant issue for us, since we only require the hands be accurately tracked. However, we did find that large jackets interfered with the hand positions.

We also found the minimum hardware on the connected machine to run the Kinect was surprisingly high. It would seem the Kinect can be quite a CPU hog.

**[Dev doc]**

The Kinect Controller Input class is the implementation of the *IInputManager* interface for the Kinect device. It registers to the Kinect that it wants to receive skeleton data, which it then uses to generate hand objects. The Kinect sends us data by means of a callback method, so we have to use a lock to ensure GetHandPositions doesn’t happen during the middle of an update.

|  |  |  |
| --- | --- | --- |
| **KinectControllerInput** | | |
| Initialize | Screen Size | Implementation from *IInputManager.*  This sets the scaling variables and attempts to connect to the Kinect. |
| GetHandPositions | **Returns** Hand[] | Implementation from *IInputManager*.  See description in *IInputManager*. |
| convertRawHandToScreen | SkeletonPoint  **Returns** Vector2 | Converts a raw skeleton point from the Kinect skeleton data to a screen coordinate. This method is just a wrapper around the Vector2 method below. |
| convertRawHandToScreen | Vector2  **Returns** Vector2 | Converts a raw point from the Kinect to screen coordinates. It applies a scaling factor in each axis and the transforms it from the raw coordinate space to screen coordinate space (defined in *Initialize*) |
| SkeletonsReady | Sender Object  Event Args | This is a callback function that is assigned when the *Initialize* method is called. It gets called once for each frame the Kinect calculates. It uses the skeleton data to generate hand data. |

**[It might be nice to have a table stating which configuration values affect this class in what way]**

**[I’ve also added a bit on gestures since I did most of that. Again, more as a reflection]**

Gestures

We had originally designed the system to use multiple gestures to manipulate and navigate it; however through advice and some minor experimentation, we chose not to do this since recognising gestures is very difficult. The Kinect has no native support for recognising gestures, so it is left to the programmer to implement these. We refined the application to use only a clapping gesture, and just use the hand positions for the rest.

We originally implemented the clap as checking if two hands are touching the same balloon, however this caused too many false positives. Users would burst balloons when they only meant to knock them, often by rapidly moving their arms around.

We then improved this method by adding additional conditions to trigger a clap. These are:

1. The hands must be moving towards the balloons (with an angle of tolerance).
2. The hands must be moving towards to each other (again, with an angle of tolerance).
3. Both hands must be moving above a certain speed.

This did help to remove the false positives; however it did make it harder to actually burst a balloon, to the point users were having difficulty popping a balloon. We added each tolerance as an option in the configuration file, so this can be adjusted easily to a value that feels natural. We found through testing that setting the tolerances to be very generous (high angles, low speeds) was best, as an occasional accidental balloon burst was better than an extremely difficult time popping a balloon.

Both of these methods use a reactive approach, only checking the conditions at the moment of impact between the hand and the balloon, however the Kinect sensor can adds extra jitter and inaccuracies to the data (as noted in Kinect section). If a jitter occurs just at the moment of impact (which it frequently does), then it will not register a clap (as the hands are not moving quite towards each other/the balloon in that frame), which is mainly why setting low tolerances helped.

A better method would be to continuously track the hands and detect when a clap is happening, as this reduces the influence of random jitter in the data stream. Alternatively, if we calculated the velocity of the hand objects ourselves as a normalised sum of the *n* previous frames, this would help reduce noise in the data.