Fat Owl

An Active Sports Project by

Sarah Koblitz
Mark Pilgram
Imen Hellali

Table of Contents

1. Game Concept

1.1 Game Overview

1.1.1 Game Concept

Fat Owl is a Hybrid Genre, 3D Game, where the player will controle an Owl while mimicking a bird in a flying movement. The player will fly through the world, explore the different aspects and sceneries. For our game the player mentality and integration plays a tremendous role, and thus the player’s body is actively participating in the flow of the game, and it’s soul is soothed by the calming yet exciting music and nature fields.

1.1.2 Genre

Fat Owl is a 3D , Runner , survival , role playing, sport game.

1.1.3 Target Audience

Fat Owl is a family friendly fun game. Each member of a family from children to elderly are able to play the game and enjoy it. The main Audience that was targeted were Children between the age of 7 and 13. Young Children are known to be more active and prefer Colorful simple games.

1.1.4 Look and Feel

The Game is designed with mostly low polygon count and flat shaded objects. Yet at the same time, as the player moves deeper into the game, it will encounter some other objects, which texture is more realistic and has a higher polygon count.

Our Main goal was to avoid having a monotonic boring experience for the player, thus we achieved our goal by changing the sceneries and playing on the displacement of the mixed simple objects and rich ones.

1.2 Gameplay and Mechanics

1.2.1 Owl Controls

The whole game consists of evaluating the skills of the player to control the Owl and delay its death as long as possible, and thus beating his own high score .

The Player stands in front of the Kinect and tries to move the Owl adequately to avoid obstacles and collect items. If the Player lifts his arms up, the Owl will consequently fly higher. If the player puts his arms down, the Owl will begin sinking and lastly, if the player tilts to the left, while pulling his left arm down and oppositely lifting his right arm, the Owl will fly to the left side, and vice versa for the owl to fly to right side.

1.2.2 Life System

The Player flies through the different fields and try to avoid crashing to the placed objects. The player has the freedom to roam around the world and choose how to avoid a certain obstacle. For instance if the player encounters the Barn, it has the choice to enter it from the upper window or the main door and exit it from the back window of back door, but at the same time it can fly around it and avoid it.

The player starts with full life counts (= 5), which are in a form of Eggs. After each stumbling with any placed assets in the game the current owl dies, the player loses one life (hense one egg), and then another owl appears to continue where the last owl left off. The Player should also pay attention to the hunger bar that is decreasing,and thus obliged to collect the items in form of Ladybug and spiders to fill it again.If the player manages to starve himself (hinse the hunger bar is empty), the current owl dies, and consequently the player loses one life and starts again with a new owl with a full hunger bar. The current game ends when the player uses all his eggs.

1.2.3 Collecting Items

The player has to collect some items alongside controlling the owl, and thus to avoid hunger as stated above. The collectables are two kind of bugs. A Ladybug and a Spider. An item is deemed collected, as soon as the owl collides with it and flies through it.

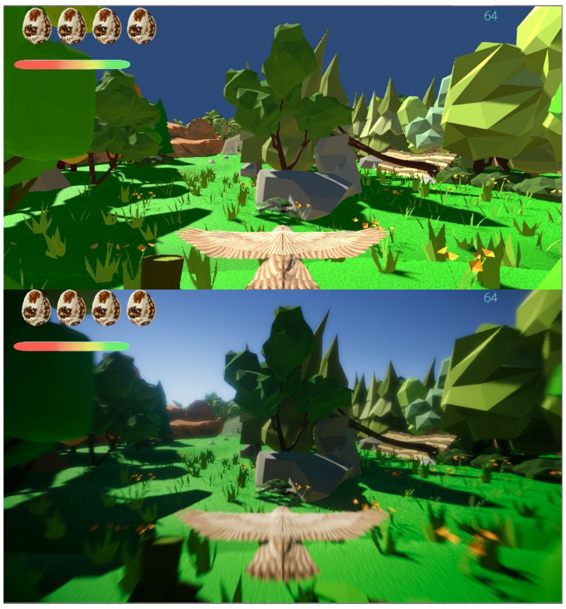
1.2.4 Menus Controls

All the buttons in the main menu, setting menu and game over menu can be selected with either a mouse click or through the action of the player whom is standing in front of the kinect. To select a certain button you hover with your left hand/arm up and down until you reach the specific button and it illuminates. To click the button you need to swipe with your right hand to the right until your right arm is fully extended.

2. Development History

2.1 Unity Setup - The HD Render Pipeline and Kinect

In the past few years Unity has come a long way. If you know where to look, it's become a lot easier to get your game to appear like it was made with an actual budget. With the introduction of the HD Render Pipeline (HDRP), effects such as Color Grading, Bloom or a high quality Motion Blur, along with a highly customizable skybox are immediately available.



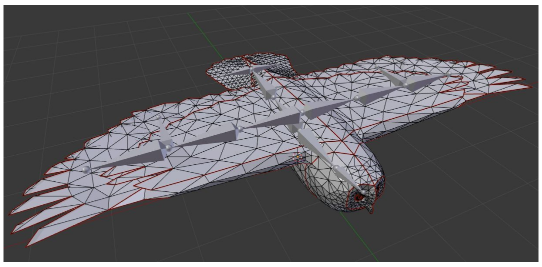
These tools were a great help to us, and allowed us to create a much more vibrant looking world, as can be seen in the comparison pictures on the right.

While Fancy visuals are nice, there was also one other thing that needed to be considered during the setup of the project: Kinect integration. For this we used the "Motion Capture and Gesture" project provided for Active Sports as a template. Since this project was created with the regular 3D project template we needed to copy all of the relevant files to our HDRP project, which ended up working just fine.

For file sharing we used Unity's built in Collaborate tool, as it has proven to be quite reliable in previous projects. With only basic Unity accounts available to us our Team size of 3 was just about small enough to make use of this feature, though there were a few times when assets had to be cut due to the 1GB cloud storage restriction.

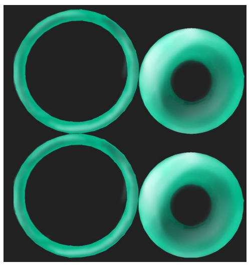
2.2 Making the Owl

2.2.1 Modeling



The first step in creating the owl was to model it. For this I used blender, since it's free and the modeling software I have the most experience with. To get things started, I searched for side views of owls on google, and once I had found one that I liked, traced its profile into a line that essentially cut the bird in half. From here the wings were slowly built out using various reference pictures. Without the perfect picture of a real owl T-Posing, this part took a bit more experimentation to get looking right. The Owl itself is mirrored along the center, so effectively only half of the owl needed to be modeled.

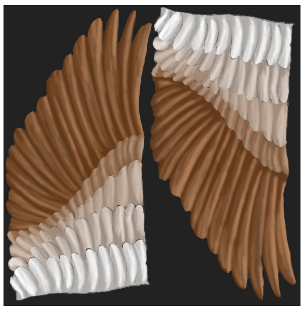
With the wings done, the face was the next (and possibly biggest) challenge. There are quite a few different types of owls with very different faces, and having never modeled an animal before the goal was to find one that looked appropriate and was fairly simple to make. The results required some tweaking, but certainly could have turned out a lot worse. From there came the more tedious task of filling the belly with triangles and finally finishing off with the tail feathers. The Owl's legs and talons were left out in the end, as they would have taken considerably more effort to make and would have served little purpose in the final game.



With the model complete, the next tasks were to create UV maps to be textured and rig the owl to be animated. For the UV maps the bird was split into multiple sections with their own material and texture map. Having one giant texture map for the owl using one material would certainly be more efficient in terms of the final game's performance (not that optimization is really considered all that much in university projects like this), but would have potentially made the process of texturing the owl a bit more complicated and given us less freedom with tweaking individual parts of the owl in the end.

The rigging process - another thing that I hadn't done before - basically involved placing bones in places that seemed logical and hoping that blender would automatically assign the right vertices to bones, which it generally did. A fun fact here: the owl's spine isn't actually connected to its wings. Had they been connected, the chances of vertices being assigned to the wrong bones in critical places would have probably been a lot higher.

2.2.2 Texturing



As the overall type of texturing and shading in the game was a mix between flat shaded and smooth shaded, in addition to simple shapes and more complex shaped objects. The Owl had to fit in the world. That is why for the colors of the owl, we chose gradients of beige, that go from white to grey to dark brown. Additionally the feathers were drawn as simplistic as it possibly could be and almost look like a kindergartener draw them, but the shading was smooth and realistic. where the creases were darker and bumps were the brighter.

2.2.3 Animation & Kinect integration

With the owl model finally being available to use in Unity it was time to grant the player control over it. The player effectively has two forms of control: controlling the pose of the owl and controlling its movement. Both are relevant to gameplay, as changing the owl's pose also modifies its collider. For the most part both rely on measuring the angle created by the two Vectors

(HandPosition - ShoulderPosition)

(SpineBasePosition - ShoulderMidPosition)

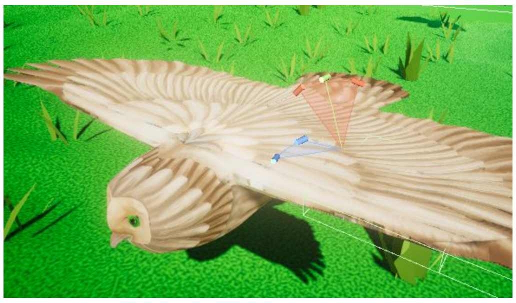
for each hand. This gives us the angle of each Arm (even if it isn't really stretched out) to the ground, which can then be translated into the angle at which each wing should be raised.

While the way to implement this pose control was quite obvious, creating a way for the player to control their movement was not. The original plan was to have the player flap their wings to fly up, tilt their arms left or right to move side to side and droop with both arms to fly down. Moving left, right and down was simple enough, with the angle of both wings available we simply needed to measure the average angle of both arms as well as the angle difference between arms to determine what the player was trying to accomplish. The flap detection on the other hand caused us a lot of trouble. Two competing solutions for it were tried, one involving gestures, the other using average arm angles & speeds as well as looking at the recent angle and speed history to guesstimate the player's desired input. This solution ended up being more successful, as it was able to respond faster and in more varied ways to the player's actions. The lift from flapping the wings could be adjusted based on the distance the player moved their wings and the speed at which they did it.

However this solution also came with many problems. Neither was quite as reliable as we had hoped for, and in testing we soon realised that flapping your wings to fly up can be very tiring. In the end the option to fly up by raising both wings was added, as this form of control - while perhaps slightly less realistic - takes a lot less effort to play, responds almost immediately to input and has no real chance of not being recognized.

As a final element of control, the player is able to temporarily pull the owl's wings together into a tight ball, allowing them to fit through tight gaps. This is triggered by moving the player moving their hands between their shoulders. When this happens, a predetermined animation is triggered, during which the player temporarily loses control over the pose of the owl. Due to limitations in unity, the animator controlling this animation has to be disabled whenever the player is given control over the owl's pose, otherwise the pose is overwritten by the animator, even if the values determining the pose aren't currently being modified by an active animation. Simply having an animation that modifies these values takes away any control your scripts should have had.

2.2.4 Fun with Ragdolls



A vital aspect of the game was to make crashing into obstacles entertaining. Unity has a lot of built in tools for creating different types of joints, with character joints being the type that suited our use case best. Unfortunately the joint system can be quite confusing, with a lot of different values to to tweak and unhelpful in-editor visualization to go with it (as seen in the picture - not all of the indicators align with the axes they control). It took several hours to figure out which values needed to be set to what to get the results we desired, but eventually it worked. When the player collides with an object, the regular controllable owl's controls and visuals are disabled, a ragdoll owl is swapped in with all of the armature's positions and rotations being applied to it. The result is a seamless transition between regular gameplay and the player character flopping to the ground.

2.3 Level Generation

2.3.1 The Logic behind the Level Generator

Like many other games in the endless runner genre, our game generates levels by placing predetermined level blocks along the path of the player. To make the spawning process of distant blocks less visible, blocks rise up and rotate towards their regular orientation in the distance, similar to how objects would appear on the horizon on a real globe (assuming the globe had a much smaller radius than earth allowing you to see the process more clearly).

This process is controlled by a script attached to each level block, which also defines which level blocks can follow that specific part of the level. The main level generator script basically just checks if the end of the level is close enough to the player that a new block needs to be spawned and - if that is the case - chooses a follow up block from the list provided by the latest, most distant, piece of the level.

2.3.2 Creating the Level Blocks

The core of each block in the level is its ground plane. Rather than having a simple flat plane, a set of slightly deformed planes was created in blender (using a displacement modifier with various textures that blender had generated for us) to give the appearance of rough ground. Each plane curves down towards its end to make the transition between level blocks more seamless. With the exception of the barnhouse, almost all of the assets we placed on these planes were taken from a variety of free low-poly vegetation packs from the asset store. We originally wanted to have more realistic trees where you'd have to dodge individual branches, but it turned out that the HD render pipeline in the version of Unity we were using wasn't compatible with the materials used by Unity's built in tree generator.

Overall though the choice of going for a more low-poly look rather than something realistic was probably for the better, as it is much easier to pull off. There are more stylistically fitting assets available with a consistent level of quality for this look and custom assets we created for the game didn't have to be overly detailed.

It's worth noting that not quite all of our assets conform to the low-poly style. All of the orange rock faces were taken from Unity's "3D Game Kit"[1], and are primarily used to hide the background skybox below the horizon when flying through the level. However this wasn't the original reason why the 3D Game Kit was chosen: The project included with this kit contained a very useful script called the "Instance Painter". With this script we were able to paint primarily small objects (grass, flowers) in large quantities onto arbitrary objects. This was needed, since our level blocks use custom 3D models rather than Unity's terrain system, so the normal method of painting these elements into our scene was not available.

[1] https://unity3d.com/de/learn/tutorials/s/3d-game-kit

2.4 Life System and In-Game UI

2.4.1 Bugs, the edible implementation

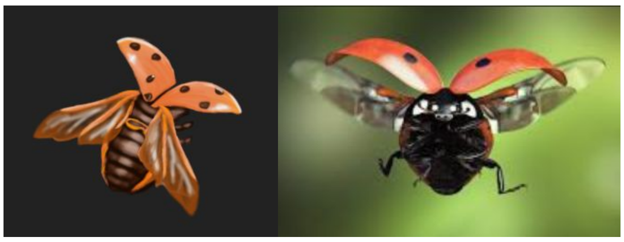
The bugs are necessary for the owl to survive a long time. When the owl is not eating it will starve and die. There are two bug variations implemented, on the one hand, there are spiders which hang on hills, branches or can be found in barns. On the other hand, there are ladybugs flying around a small circle, which makes it hard to eat them but they are more frequent then the spiders, which are stationary. For the spawn of the bugs there were placed spawn points to make it easier to control the bugs. The first solution was a random-on-screen-placement but it turned out as very unstable, as described in point 3. b.

The next and final solution contains the first solution by placing the bugs randomly. Not random in the point of the amount of bugs but random in the fact if on that specific spawn point should spawn a bug or not. That makes it more unpredictable where the next bug spawns although the spawn points are fixed.

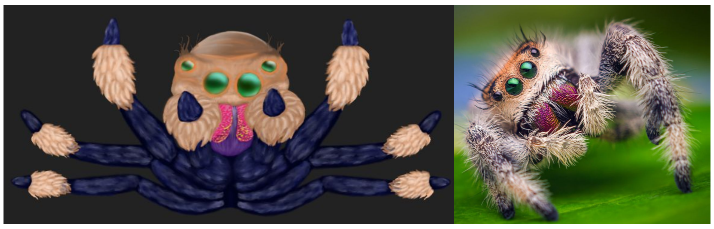
2.4.2 Bugs, the edible Art and Design

Our Bugs were sprites from our own creation and hand animated. As mentioned in sectioned one, to obey to the overall design and art style of the game, our bugs were drawn differently and animated differently.

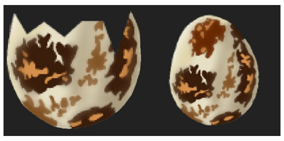
The Ladybug looks more realistic and has a more complex movements, where it flies in a circle and flaps its wings. it was drawn in a realistic ways and smooth shaded. The colors were adapted to convey to the real world found ladybugs and darker shadows in opposition to lighter shadow were added to give a depth illusion to make them look like they are 3D objects.



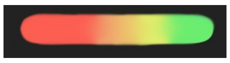
The Spider on the other hand, although it was smooth shaded and was given an illusion of depth through the coloring, the movements were simple and looked like stick figures movements. The spider was inspired from a real spider that had vivid colors and a cute appearance, to avoid having our playing feeling disgusted through the game. It was also taken into consideration while creating the collectables, the people who have specific bug-phobias.



2.4.3 Life system and UI



As the whole game was based on an owl, we chose to represent our lives as eggs. Eggs refer to a new life where the owl originated from. when the lives of the Owl drop to 3 we decided to show half eggs instead of none. Half eggs as opposition to numerous games they do not represent half lives but rather, we wanted to demonstrate a hatching action after each death where a new life will rise. Yet leaving all five of the eggs and make them as half shells will crowd the screen and deviate the attention of the player.



The Hunger bar, which was a slider, was also made in colors to adapt to the young and energetic feel that our game give off.

The Hunger Bar value is decreased dynamically with each frame by a small amount. The Amount of decrease was adapted many times to make the level of difficulty adequate to the player's skills.

Each time the player stumbles upon a bug the hunger bar value increases by a proper amount. Because it may be hard to catch the bugs, the reward after collection was made higher, so the player won't feel distressed.

As stated above the player has to avoid obstacles placed in the world and as hitting these objects is fairly easy, we first started the game with a full life (5 Eggs) and a full hunger bar and a plain scene (start level block), to allow the player to get used to the movements and then the real blocks come afterwards.

Whenever The Owl hits an obstacle and dies, the UI would be updated and live count decreased, and a new owl will spawn instead of the previous one which takes its last position and adds to it in terms of depth (z axis), because the player needs to go past the object that made it dye, and another life basically represents another chance, that is why we help the player pass this difficulty and try again when it comes again. The player was also given a full hunger bar. We obviously didn't won't for the player to worry about dying again as soon as it came back to life.

2.5 Sounds

2.5.1 Music

The music-making was challenging, because every team member wanted another theme or had other expectations of the feeling that should be transmitted. The first try was to compose an own theme for Fat Owl with the sound software ‘Reaper’. In the first weeks some sample sounds were made but none agreed with it. That threw us back to the beginning and we decided to choose from already bought soundtracks, called ‘The Ultimate Game Music Collection’. There are around three hundred different soundtracks. One team member has listened to all of them to make a small selection of which the other members can choose. Finally, all agreed with three soundtracks which were prepared afterwards with nature sounds like bird twittering to make a great ambience to fly in and one without nature sounds. The preparation was also made with ‘Reaper’. All soundtracks were used in-game, there is one in the start menu, one in the settings, one is used as the play sound and the sound track without nature effects is placed in the game over screen.

2.5.2 Sound Effects

Sound effects are very crucial when bringing a game to life. There is also a already bought selection of around five thousand sound effects, called ‘Universal Sound FX’. From there we took the crash sounds and the eating sounds. The crash sounds are a variety of three different sound effects to give more diversity. They are played randomly when the owl collides with obstacles.

There are also 8 different owl sounds, which do not belong to the sound effect collection, because they are recorded on our own. The owl sounds are made by blowing air in folded hands and recorded with a professional microphone in a home studio of one team member. The sound effects are handled in-game with an own script respectively in the life controller due to the crashes or the eating behaviour.

2.6 Menu

2.6.1 Main Menu

As obvious as it may sound, the first encounter with a game is very important, and it heavily influences the mood and the feelings that a player will have while starting the game. That is why the first thing that a player will see needs to be appealing full of life and represents to a certain extent what the whole game is about. That is why we chose our self-made 3D model to present as first thing to the player with some moving clouds to avoid boring liveless scene and added our bugs. The main menu is also controlled by the player through gestures although buttons can be selected with a mouse click, but it was crucial for us, to make the player at ease. In addition to that we tried to implement some intuitive yet ease and fast to code control-gestures, and that is how and the idea of selecting the button with left hand and clicking on it is executed through a right hand swipe to the right. (this action is similar to all the other menus).

In The main menu before each button click was recorded, a delay time (waiting for seconds) was added. The reason for that was, that the player may be slower in conducting actions and changing the position of its hands, thus may lead to selecting a different button than the desired one.

As the hand of the player is detected elsewhere in the scene, the delivered position was adapted to fit that of the Canvas and than the whole canvas was divided to three section, where each section represents a different button.

When the hand of the player moves within that specific section, the corresponding button is selected.

Independently from the left hand, the player has to swipe his right hand to the right to invoke the selection button method. That action were implemented in a simple way where if the right hand enters a specific margin it will be deemed as accomplished.

2.6.2 Settings Menu

In each game there must be a setting menu, and a setting menu is also no exception to a main menu. Although it won’t contain too many details concerning the game, it still had to look more joyful and livelier than all other menus, because it simply contains the most important information to the player.

Here we display the highest score the player has ever made.

the Player has also the opportunity to adjust the volume of the whole game with gesture through this menu.

The volume bar is just a slider that takes in a value between one and zero. As stated above all selecting and clicking/changing action within menus were conducted with left hand and respectively right hand. It was only natural to keep that flow and make changing the volume bar action with the right hand, and thus by fitting the position of the right hand into a small range, namely between 0 and 1 and dynamically register and change the volume of the Game. When the player slides his left hand away from the range of the volume (volume disselected) the last state of volume will be registered. All registered values were made with playerprefs.

2.6.3 Game Over Menu

When the player finally runs out of lives and dies, it will be redirected to this scene where he will get the choice of retrying the game, go to main menu or quitting the game. The score collected in this play session will also be displayed there.As mentioned above each element that we needed to save, keep and then display again were registered within playerprefs.

As the Game Over scene depicts the end of a game and a loss, it had to look gloomier and sadder than the rest of all the scenes. That is why it was designed to look dry, simplistic with minimum signs of living creatures (trees and grass) but at the same not deviating a lot from the overall feel of the game. All the feeling, we wished to deliver were amplified with the sounds and choice of music that were added.

2.7 Day & Night Cycle

The day and night cycle - planned as a nice to have feature - was a late addition to the project. Its implementation is quite simple: Using an animator the celestial bodies are moved across the sky, with their light intensities being adjusted in much the same way. Some skybox properties are also modified as time passes, though this wasn't possible through the animator. Instead an accompanying script checks the animator's animation playback position and uses this value to evaluate various animation curves, that then determine the skybox properties. Since the night cycle was so dark, we added an additional light that follows the player, lighting their path from behind at night.

At first this feature only worked as intended when playing the game in the Unity Editor. When playing the game as a standalone program, light sources would refuse to change their brightness. As a result, the sun could be seen traveling back to its day starting position during the night cycle while the moon was invisible.

This was eventually fixed by animating more parts of the cycle through script rather than through Unity's own animator, as well as animating the Sun to also be the moon, rather than having two separate light sources. There's no logical reason why these things didn't work beforehand or why this fix - that was basically made by messing around with a bunch of things until something actually worked - should make any difference, but it did.

2.8 Making the Trailer

2.8.1 In-Game Tools

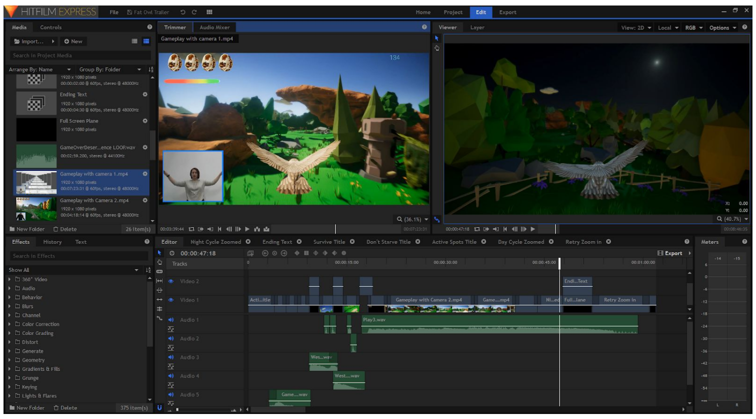
For the creation of the trailer, several tools were added to allow us to get the shots we needed that weren't from regular gameplay. For starters, additional camera modes were added. The game can be paused to reposition the camera and then unpaused to show the game running from this custom camera position. If needed, the camera can be unparented from the player character to capture a static view of the gameplay.

The day and night cycle could often get in the way of recording the action at the perfect time of day, so at the press of a button it can be toggled on or off. To make recording scenes of the bird starving in mid air faster, we added the ability to make the owl starve almost instantly at the press of a button. Upon death, the camera usually follows the ragdoll of the owl, but this wasn't necessarily always desired, so an option was added to disable this behaviour. Finally a toggle for slow motion (25% game speed) was added, because everyone loves a bit of slow motion.

2.8.2 Recording challenges

Being a Kinect game, we wanted to feature people playing the game in at least a part of our trailer. Our game however never explicitly displays the Kinect's viewpoint to the player, the player's pose is only indirectly displayed through the rotation of the Owl's wings. Fortunately there was a fairly easy solution to getting a camera feed displayed in our recordings. Using the recording software OBS, we could simply overlay footage from our Laptop's webcam over the recording of the screen, making it look like we had a feed from the Kinect in our recording.

2.8.3 Video Editing



To edit the trailer we used Hitfilm Express, as it offers pretty much all of the features you need in a project of this scopre for free, such as multiple simultaneous video and audio tracks, compositing & animation tools and special effects. We did end up using a trial version of the "Hyperdrive" visual effect, which is why if you pay attention during the "Survive" title part of the trailer you will see a giant Hitfilm watermark in the background of that specific scene. Since we really wanted that effect and the watermark wasn't too distracting (only appearing in that short section of the trailer), we decided to leave it in and not pay the ~15€ for the VFX pack that it was a part of.

3. Development Issues

3.1 Making responsive controls with Kinect

We had originally intended to use gestures more in our game to control our Owl, however in our testing these gestures rarely delivered satisfactory results. Our conclusion was that gestures come with too many inherent problems to be practical in our somewhat fast-paced game. For starters they introduce delay: A gesture can't be responded to until it is complete. This creates a delay between the player's input and the action in-game, causing a disconnect between the two. Gestures also run the risk of not being detected properly or even interfering with other actions. Having inputs be inconsistent is probably the fastest way to get your player frustrated with your game.

For a Kinect game you want your input to match the player's intended input as closely as possible with minimal delay, which is why we ended up with a system that basically just turns the player's arms into a giant analogue stick - the current average arm angle for both arms determines its vertical position, the angle difference the horizontal position. This allows us to continuously and reliably use the player's most recent input with no room for error beyond the Kinect itself not recognizing the player properly.

The only gesture that was left in the game allows the player to pull himself together to fit through tight spaces. Since this was fairly fast and easy to detect (we just check if the player's hands are between their shoulders), doesn't interfere with other controls and has little relevance in the core gameplay it didn't cause too many issues.

3.2 An excessive amount of bugs

It's not a feature, it's a bug. This happened in the first version when implementing the spawn of the bugs. That was quite funny, because the 'bugs' now were actually bugs. The problem was, that each frame started a new coroutine which actually should spawn one bug each five seconds. Due to the each-frame-calling there were a massive amount of coroutines which spawns randomly animated ladybugs on the screen. This causes a high amount of bugs therefore the screen was full of them. At this time the picture was taken. To solve this problem and to handle the bugs in a better way, we decided to do this with fixed spawn points as described before.

3.3 Differences between Unity Builds and playing the game in the Unity Editor

For reasons that are still unclear to us, things that worked perfectly fine when played in the Unity editor don’t always behave in the same way once you build your game. In our case it would appear that animating directional light properties using the Animator - while this works just fine in the editor - may stop working when playing a game build. This may also be tied to whether directional lights are defined as the sun for the skybox, it’s hard to say. Eventually messing around with various things did give us a working build, though it’s hard to say what specifically ended up changing the result. With building our game taking as long as 10 minutes, the trial and error process for fixing this problem was quite tedious.

4. Task Distribution & Schedule

4.1 Task Distribution

4.1.1 Sarah Koblitz

It was exciting to work with a medium I have never worked with. In the first two weeks we learned together how to control the kinect and afterwards we divided the tasks that have to be done. Because it's about programming, each of us had his part in the code. My task field was to implement the bug behaviour. That sounds very common but I made several mistakes which can be avoided in future. We all are learning with the mistakes we do. After implementing it correctly and working my main focus laid on the sound. It did not worked out to create a soundtrack for the game on my own, because I couldn't combine all the wishes of the group members. That's why I customized already existing soundtracks with the sound of nature. The owl sounds ('hohoo') were recorded by myself in my home studio. They are created by blowing air in folded hands. Also the integration of the sound in the game was made by myself.

4.1.2 Mark Pilgram

For me the goal of projects like this is to try new things. In this case those new things included modeling, rigging and animating a living being (our Owl), as well as creating a ragdoll for this model. Making our player character controllable with Kinect was a challenge we all worked on, and as mentioned previously we ended up trying two different ways of translating Kinect input into player character movements, with my solution being the one to make it into the final game. I had previously been looking to create a game world where distant objects would fold up into the player's view rather than appearing out of thin air, so I also used this opportunity to make the system for spawning and managing level blocks. This doesn't include the spawning of bugs within those level blocks.

Less Interesting tasks of mine include modeling the Barnhouse (also used as a Greenhouse in-game) as well as generating various ground planes in Blender, then later populating them with objects to create the level blocks we have in our game. I also added the day and night cycle to the game, though most of the time here was spent on getting it to work in our build version of the game rather than creating the day and night cycle itself.

Finally having previously used Hitfilm in various school projects I was the one control of our editing software for making the trailer, though decisions for the content of the trailer and how it was to be put together were made as a group.

4.1.3 Imen Hellali

Developing the basic idea of the whole game. Also majorly deciding on the game mechanics, yet afterwards refactored alongside other members. Deciding on the life system and hunger system, then implemented the basic functions, behavior and in-Game UI updates, then Sarah took off to add the bug-eating and adjust it. Also developed the reward/scoring system. Putting in place and implementing player interaction and controls for all the other scenes (Main menu, Settings, Game Over) excluding the play scene. Drawing the 2D sprites for the bugs, texturing the owl and barn, eggs as lives, hunger bar, and in each scene UI needed Image (button sprites). Hand animating the bugs (traditional animation with different pictures each frame).

4.2 Meeting the Milestones

4.2.1 First Milestones

Below are the Milestones we wanted to reach when we started scheduling and designing:

➔	20.11.18:	Implement Kinect controls
➔	27.11.18:	random level generation from a selection of blocks
➔	04.12.18:	Life & hunger systems
➔	11.12.18:	Main Menu, Game Over screens
➔	18.12.18:	Create more level blocks
➔	Holidays:	Finish up things that took longer than expected
➔	08.01.19:	Bonus Features
➔	15.01.19:	Bonus Features
➔	22.01.19:	Bonus Features
➔	29.01.19:	Presentation & Game Demonstration

4.2.2 Actual Milestones

These are the actual dates when the Milestones above actually worked and it’s obvious that the deadlines moved backwards. But not as much as expected.

➔	21.11.18:	Implement Kinect controls
➔	29.11.18:	random level generation from a selection of blocks
➔	11.12.18:	Life & hunger systems
➔	22.12.18:	Main Menu, Game Over screens
➔	17.12.18:	Create more level blocks
➔	19.12.18:	Ragdoll
➔	Holidays:	Finish up things that took longer than expected
➔	08.01.19:	Sound integration
➔	09.01.19:	Scoring System
➔	12.01.19:	Day and Night Cycle
➔	24.01.19:	Trailer
➔	29.1.19:	Presentation & Game Demonstration

Summarized, we were almost in our time schedule and we can be proud of what turned out in the end. There are some Bonus Features that were cut out, but compared to what we have implemented the features would have disrupted the time frame.