Turkey Jump!

Lilly A. Helmersen - Michele Romani - Zack Wilson

Inspiration

For this project our goal was to experiment with new interaction techniques inside well-known contexts. Video Games are the intersection point between science and art and we believe they perfectly fit into the spirit of Human Computer Interaction.

We decided to revisit a very popular game, *Flappy Bird,* an *endless* platformer that became famous a few years ago thanks to YouTube videos of players struggling while trying to challenge the simple but difficult mechanic of the game.

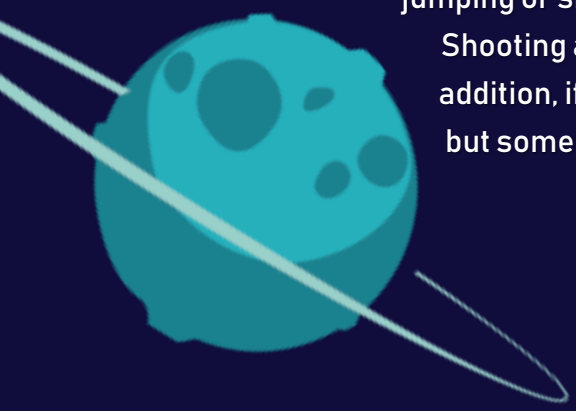
All of these videos have in common the fact that players used to scream very often while playing due to the difficulty, so we decided to turn these screams into a new way of interacting with the game.

Add some asteroids and aliens and here you are: **Turkey Jump!**

Functionalities

Turkey Jump is a platformer style game built with Unity where the user controls all the gameplay using their voice. The game play involves standard platformer rules and actions. The player's mission is to save Kevin the SpaceTurkey from being eaten by space farmers during the Lunar Thanksgiving.

We augmented the basic platformer gameplay by adding aliens that can be avoided by jumping or shot with lasers, which the user collects through game play. Shooting aliens scores points, while just avoiding them does not. In addition, if the player collides with an alien, the turkey doesn't die, but some points are stolen.



Implementation:

The structure of the implementation follows the PAC model, using different controllers for specific functionalities for better *separation of concerns* and *modularity*:

GameController: defines the rules of the game and changes the state of the game according to them. As entry point, it initiates an *instance* of the game every time *Start()* is called.

PlayerController: is the controller and the view of the player, it updates the player model according to input

VolumeController: is the controller that manages volume functions reading input from the microphone. It processes audio and apply a *transform function* to translate it into movement of the player.

VoiceRecognitionController: is the controller that manages voice functions reading input from the microphone. It processes audio and stores it as game commands. It has been implemented using a **KeywordRecognizer**

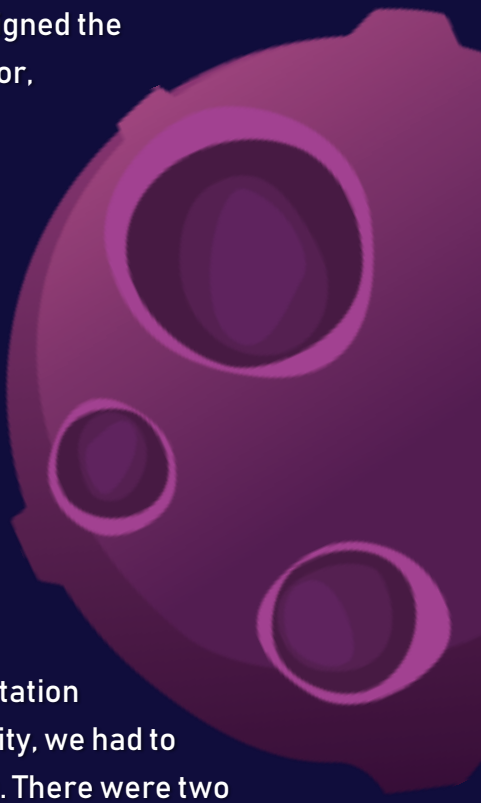
AlienSpawner, CraterPool and AsteroidPool: spawns respectively aliens and obstacles. These classes allows the *endless* mechanic of the game.

During the whole development process we have been working with many different tools such as Git and **GitHub** for source control and Adobe Illustrator to build our assets, allowing us to coordinate as a team and take advantage of our previous backgrounds.

Check also our GitHub Page!

Challenges:

Seamless voice to position control: The key functionality we implemented beyond the game play, was to map all the game controls to the user's voice. This takes the form of voice commands for more complex tasks, and volume analysis for jumping. Using their voice, the player makes the player jump, over aliens and obstacles. We created this using a volume analyzer, and so mapped substantial changes in volume to jumps. However, this actually was not a very intuitive way to use voice control. This design challenge, to make voice control fluidly map to 2D movement controls, was the most challenging part of the implementation. In the end, we redesigned the feature to use a continuous jump, so the longer the player is loud for, the more the turkey jumps. This correlates in a more natural way with voice than individual jumps based on brief shouts and yells.



Voice recognition control: The second functionality of voice recognition is the actual voice commands the user uses to start the game, restart, and shoot (this also includes an easter egg where the command 'die' ends the game – so anyone near the player can end the game). We used the Windows Speech Recognition API to create a list of commands that the program can accept. While the lag of this API made it too slow to use this for jumping, it is quick enough that it works for starting and restarting, and even shooting.

Memory Management: Another important aspect of our implementation was memory management for Unity. Because we built the UI in Unity, we had to keep track of all the components we were spawning and changing. There were two main ways we did this, first, optimizing the way obstacles were created, saecond, keeping track of components we spawned. In order to optimize the obstacles, we used the pooling strategy. We spawned five obstacles off screen at the start of the game, then simply cycle through these, placing the last one – which is off the screen behind the player – in front of the user again. To keep track of aliens and projectiles we spawned, we added destroyers than follow the game off screen, and destroy any objects that escape the game play area.

Engagement: To make the game play more convincing (and fun), and to provide better feedback, we added animations to the turkey for running, jumping and dying. The animation was added using Unity's built-in sprite reader and frame animator, using a State machine to determine its current state. With the animation matching the Turkey's different state, running, jumping or dead.

Simplifications

During the development period, also due to time constraints, we realised we could not implement everything we imagined in our mind in the beginning. These simplifications are worth mentioning:

Voice commands - Due to the latency of our speech recognition API, we were forced to only use speech for actions that were less or not at all time dependant.

Multiplayer - Perfecting the game play took too long to allow us to develop a new multiplayer mode. Any imperfections were so apparent in a game based on speed and precision, especially when using voice controls which are inherently a more imprecise method of input.

Balance - This kind of games may be really difficult to balance, either it was too difficult or too easy. We managed to reach a decent level, but we realise the game is still in its early stages and many things can be improved.

What's the use

Our project was a way to experiment with voice as an input method, by making a (hopefully) fun game. The point is to have fun! However, the voice processing we implemented could be used in other contexts. As an easy illustration, we already made two games (the simpler Flappy Bird and the more complex platformer), so we imagine a complete "voice arcade" where these controls are applied across a wide range of games, in creative ways. Beyond gaming, our voice to control mapping could be used for controlling anything with one dimension. Also, the voice control has a way of making the game interaction more public, which in turn makes for a more communal, inclusive gaming experience (based on our tests while developing this).

Our journey from Earthly bird to the Space Turkey has been long and full of mysterious bugs, but it's not over. Many are the functionalities we are working on: multiplayer, social interactions, ranking and much more.

In the meanwhile...

There is no noise in space

so make your own