**Project 2: Flappy Bird Interim Report**

## Viktor Neshikj, Ian Kuik, Benson Cho

## **Game Strategy**

Flappy Bird is a simple game. It involves players controlling a bird’s y motion. A click of the mouse should allow the bird to rise and continue flying while dodging pipes and not touching the ground on the screen. Each successful set of pipes dodged results in one point. The player is aiming for the maximum amount of points.

The game will be playable in two modes. Training and normal operation. In normal mode, the player will control the bird while avoiding pipes. They will be given three lives. One collision with the pipe or the ground will cost them a life. Once all lives are finished, the game is over. In normal mode, the game’s difficulty will scale with score. After a certain amount of points (to be determined when testing), the pipes will come faster and be closer together. This will become more demanding for the player to dodge and will add a level of challenge to the game. There will also be power ups available for the player to pick up. These will include (but are subject to change, as we haven’t implemented them yet) invincibility (where if collisions occur, they will not end the game for a limited amount of time), a health pickup (where if the player picks one up and have less than three lives, one of these lives will be refilled).

In training mode, the same basic game flow occurs, except that there is no difficulty scaling with player score. This mode is purely for practice. Power ups and lives will be enabled.

We will also allow for character/ sprite swapping via switches on the FPGA.

#### **Design Specifications**

In order to play this game, there are certain design specifications we must adhere to. Firstly, it must be displayed through a VGA interface. This means that our game must adhere to the 640x480 screen size. The screen should also refresh at 60 frames per second, requiring a VGA sync with clock period 16.67ms.

The game should be compatible with a DE0-CV board and will be implemented in VHDL.

In order to fulfill our game strategy, we will need to implement four states in a FSM; game\_start, normal\_mode, training\_mode and game\_over.

#### **Planning**

As stated above, we will need a four state Moore FSM that will keep track of the game state and will have inputs of bird collisions and mouse clicks to change states.

At our core, we will need a bird component (similar to the bouncy ball) whose Y-position will increase on mouse-click. It will need to fall when the mouse is not clicked and have its own speed and acceleration. It will also check for collisions between pipes, ground and power ups.

To control the bird, we will need to create a mouse component that sends mouse clicks to the bird in order for it to rise.

We will need a pipe generator and an array of pipe configurations to be accessed/ indexed using a random number generator to generate pipes in a random order and provide challenge for the player. When a bird collides with a pipe, a collision signal should be passed into the FSM to trigger a state change. If the bird passes a pipe, a signal should be sent to score in order to increment it. This will require its own component.

We will need a component to display text. This will include the score, game over messaging and our game title.

The background and ground will also need their own components with output to VGA\_sync.

Power ups will also need their own components and may need to input into pipe generator or FSM to change states/ implement their ability.

We will also need BCD to seven segment converters to display our score on the FPGA.

And finally, we will need VGA\_sync to display all of our components as well as a multiplexer with priority to display our bird, pipes, score, background, ground and power ups in the correct order (with bird on top).

### **Finite State Machine**

We have decided to go through with a Moore architecture for our finite state machine. The inputs of our FSM will be ***clk\_in*** (25 Megahertz clock signal), ***collision*** (indicates whether the bird has collided with the pipes or ground), and ***mouse\_click*** (input from the mouse). The output of the FSM is a two bit signal (***state\_out***) representing the four different states which are responsible for the modes of the game.

#### **Game States**

When ***state\_out = “00”*** the game is in its ***game\_start*** state. In this state the input of ***mouse\_click*** is checked over a debounce period. Should the mouse be clicked during this period the FSM will transition into ***normal\_mode*** state, otherwise the FSM continuously awaits for input counting up to the debounce time and resetting each time until the input is received.

When ***state\_out = “01”***  the game is in ***normal\_mode*** state. In this state the FSM if any collisions have been detected, transitioning into ***game\_over*** state if a collision occurs. If no collision occurs the user is able to play the game normally as intended.

When ***state\_out = “11”*** the game is in the ***game\_over*** state. Again in this state the FSM will await for an input to be received from ***mouse\_click*** in a specified debounce period. When the signal is received during this period, the FSM will transition back to ***game\_start*** where the user can start playing once again.

When ***state\_out = “10”*** the game enters ***training\_mode***. This state has similar functionality with the ***game\_start*** state**,** the FSM checks if the board has collided with pipes or the ground.

### **Block Diagram**



