Kirby vs King Dedede (super ultra laser attack game)

Daniel Gerber
Tynan Smith
Output Submodule:

If(rgb == red, reddone == 0)
  redx = x;
  redy = y;
 reddone = 1;
If(rgb == green, greendone == 0)
  greenx = x;
  greeny = y;
greendone = 1;
If(x == 1024, y == 768)
  ready = 1;
Else
  ready = 0;
If(x == 0, y == 0)
  reddone = 0;
greendone = 0;
Gesture Recognizer

### Segment Encoding

The segments are encoded such that the more similar the segments, the less the hamming distance between the encodings. This is used in our simple heuristic for computing score to take into account misinterpreted segments. Additionally, special encodings indicate the start and stop of a gesture, that way if we get as input many segments fewer or more than the length of the gesture we can take this into account in our score.

### Segment Determiner

Looks at the displacement between successive points, if they are far enough apart then count them, if they are not see if the gesture has ended. Every few points check the total movement and determine the segment, if we have enough of the same segment in a row, send this segment to the pattern determiner.

### Pattern Determiner

Each gesture has its own module with a ROM storing the list of segments expected for that gesture and a register keeping track of the number of segments seen so far in the current gesture. Each time a new segment comes in we compare it to the expected segment (as well as previous and next segments in case one was dropped or added) and the score is updated. The maximizers update the current best pattern and if a stop segment is received we let the final score update and then compare it to a threshold. If a pattern matched sufficiently it is sent to the game logic.
The gesture recognizer analyzes the coordinates of the laser in real time, decomposing it into straight line segments. These segments are then analyzed in real time by modules representing each possible gesture, simple heuristics determine the score and are used to pick the matching pattern.
Game Logic

Input
● Red and Green coordinates, gesture and speed
● Keyboard input used to control players
● Vertical sync from the video output module

Output
● The position and size of every sprite

Registers
● Registers to keep track of the position and speed of every sprite
● Registers to keep track of player data (health, state (jumping etc.))

Computation
● Movement, and animation (changing sprites) of sprites – signed addition
● Collision detection for sprites – comparisons – signed addition, possibly multiplication for circles
● Simple physics (gravity, friction), counting etc. - signed and unsigned addition

Time
● Each new frame will be computed while the current one is being displayed with the new outputs stored in registers. When the vsync indicates that we are no longer drawing the current frame then the new outputs will be sent to all the sprites. We will have an entire frame to compute the next frame and sense the logic will be fairly simple and can be done in parallel we foresee no issues with completing the computations in time.

Scheduled Completion
● Upon completion simple functional versions of the video analyzer, and gesture recognition module, we will begin to work on this module. We will also update the related modules as necessary (add sprites, gestures etc.) until the game is really really awesome!
**Schedule**

**Nov 9 – 15:** Tynan: Complete testing of verilog gesture recognition modules, Dan: complete initial detection of one laser dot on white paper

**Nov 16 – 22:** Tynan: Implement gesture recognition with existing video system, Dan: complete video system to work with two dots on screen

**Nov 23 – 29:** Tynan: Finish deciding on overall game mechanics and begin game logic structure, Dan: Begin work on sprite and video output system

**Nov 30 – Dec 6:** Tynan: Finish game implementation, Dan: finish sprite implementation and all sprites

**Dec 6 – Dec 8:** Both: Finish writing up paper, tweak game as necessary and add awesome features, play game and celebrate being done