Dino Game Process

Here is how I made the Dino game.

First, I had to figure out how to use the supplied JackCompiler, so that I could test my code as I went (since I had not written a compiler myself yet). To do this, I had to:

cmd into the nand2tetris > tools directory

enter the following command: JackCompiler ..\projects\09\List

Then I began working on the class files, starting with TREX.jack.

In the bitmap code generator, I made a simple Trex in four different positions: two to alternate for running, one of which also works for jumping, and two for crouch run positions. I then made a Main.jack and function main() to try out the bitmaps and test positions for the Trex. At this point, the bitmap rendering functions involve many calculations using the location on screen to determine the RAM locations, so I think I might optimize that later, but I’m not sure; it might be nice to have the variable location placement so that I can use the same functions for different positions of a jump.

After some experimentation, I settled on having the Trex position be at 7300; this gives users plenty of time to see obstacles and also is a little above the bottom of the screen, so I will be able to add ground.

Problem: the 16 by 16 bitmap was far too small. After looking through the code and playing around with FlappyBird (url), I decided to do something similar, by rendering objects in columns of 16-pixel width, but with arbitrary height. If something needs to be wider, it is simply rendered with two columns. For each column, I can use only as many poke commands are rows, rather than poking 0 into already white space. So now I go back to the bitmap editor and start working on multi-piece graphics. I also am starting to put the bitmaps in a separate class, called Graphics, which will be cleaner code.

Since the renderings are bigger and in chunks, I further split the render functions to allow for re-rendering just the legs, for running, rather than re-render the entire Trex.

It’s been three hours and I still do not even have the basic Trex regular graphics done.

I added a tail as a third column. At this point, each chunk gets its own function and there are functions that render the different chunks together. I think it will be most convenient to have the feet chunk position as the main location for the dinosaur and offset the head as +1 word (right a block) and -512 words (up a block) and the tail as -1 word.

After three and a half hours, I think I’m finished with the Trex. It uses the same feet and tail for both standing and crouching, and two feet modes that can be switched back and forth each game clock cycle to give the impression of running without re-rendering the entire dinosaur. If we want to re-render the entire thing, however, we can, just by using the drawStand1 or drawCrouch1, which we will likely need to do for jumping and switching between stand and crouch mode.

Edit: Now, there is only one drawStand and one drawCrouch, and since both positions use the same feet, I’ve refactored drawStandFeet1 to simply drawFeet1 and same thing for drawFeet2, so Trex will simply draw such a main position and then use drawFeet1 and drawFeet2 to run.

Things I learned Day 1

* 16 by 16 pixels is far too small for a significant player sprite. The height can easily be extended with more poke commands, but to increase the width, you have to add another column, since the screen works in 16-bit words. Moving parts are most easily done in the same “chunk” of 16-pixel width, and doing so allows that part of the character to be updated without needing to update the other parts of the character.
* Locations on the screen are memory array locations with base SCREEN = 16384. So, a pixel tall row can be stacked on top of another with location -32 since there are 32 words in a screen row. Then a 16-bit chunk can be stacked by adding or subtracting 32\*16 = 512. To move to the right or left 1 chunk, use +1 or -1 respectively. All this should have been obvious from the discussion on the screen from week 4 of part 1 of the course, but it has been a while since I did that and had forgotten.

Day 2

Now I’m starting to put together a DinoGame class to run the game, and will try to set things up to get a jump on space bar / up arrow click.

Ok, so after maybe 1-2 hours, I have a pretty reasonable jumping mechanism. Of course, I had a few glitches, some of which I documented, many tries that I didn’t. Right now, I’m using a downward velocity number which gets increased each game cycle, just as gravity increases downward velocity in real life. Then, to update the Trex position, I simply find the location that is previous\_location + (downVelocity \* 32), so if downVelocity is 5, then the new location becomes 5 screen rows lower than before. I then check to make sure that the new location is not below ground by comparing it with initial\_location. If it is below ground, I simply reset downVelocity to 0 (like the ground pushing up and counteracting gravity), and set the location to be simply the initial ground state location. Whenever the Trex is on the ground, it switches its feet position each cycle.

To implement this jump without clearing the screen in between frames, I have a separate drawJump, which is pretty much like drawStand, except that it takes also as input the number of rows that were “jumped” and erases the screen for those rows in the dinosaur column. The problem right now is that it clears all but one row each time (probably some off by one error somewhere), so I need to fix that [edit: it turned out that I had been clearing rows above the Trex when it was going up and rows below when going down, while the opposite was intended].

Things I learned

* To give an object “physics-like behavior”: keep track of a velocity vector; then on commands like jump, run, etc., update that velocity vector accordingly (checking to see if the object is on the ground or in a state in which the move is valid); then every game clock cycle, update the player’s position using that velocity vector, and also update the velocity vector with independent physics (so take into account the forces of gravity, or of solid surfaces and the way they must change the current velocity in a time step).
* In DinoGame.run(), use a loop that constantly updates the different objects on the screen, then does Sys.wait(delay) for some specified delay that controls game speed.
* Not many “breakthroughs” today, but a lot of trial and error that sort of led to some results that show themselves better in the code than I can describe here.

Day 3

Today I worked on implementing crouch. After some consideration of how to represent whether the player had been crouching, just started crouching, is not crouching, etc. I settled on a simple integer number that represents the case. The different cases are important, so that if it continues crouching, I can simply swap feet positions rather than re-render the whole thing, and I need to know what things to erase (I have to erase the standing head when entering a crouch and erase the crouching head when exiting from a crouch). The code that resulted is not super elegant, but it works, and does so pretty efficiently, so I think it’s good.

There are four main things left to do in the game: add the additional objects (cacti, Pterodactyls, ground, and clouds, which will go at a different speed), generate these objects in a playable, yet seemingly random way, figure out if the player has collided with a collide-able object, and add a score tracker.

I’m going to start by making a single cactus sprite and try to get that to scroll across the screen, learn what I need to about handling collisions ect. with it, then add the other sprites. In the FlappyBird game, the Game class maintains three instances of the pipes, and the pipes themselves update by resetting their cords to the right side of the screen when they go off the screen on the left (and they also change their hight randomly on this update). I might try something similar, though the cacti will have to change their drawing and collision box as well, to enable multiple types of cacti on the map. At first, I think I will consider a cactus’s collision box to be the rectangle with the same height and width as the cactus. Since it is easy to check the interlap of two rectangles, I can then treat the Trex as two rectangles: head and body (ignoring tail collisions), and check both for collisions with the nearest cactus each cycle.

The FlappyBird game uses rectangles for the pipes that scroll across the screen and that seems like it would be easier than using sprites, since the OS has convenient rectangle drawing commands, but rectangles will not work for Pterodactyls or clouds, so I will have to do something using bitmaps. My current idea is to come up with 16 different bitmaps for the same object (1 for each index in a screen 16-bit word, then cycle through those 16 bitmaps, and every 16 clock cycles, changing the actual location of the cactus. I could actually do 8 bitmaps instead, and just have the cactus move two pixels each time step, or 4 and move four pixels each step. I will try these different things with a simple rectangle before putting the effort in to the cactus bitmap drawings.

After maybe an hour, I have this well and working. It turns out that the 8 frames, moving the triangle 2 pixels to the left each time works well and is quite smooth, so I will also stick with 8 frames for the Pterodactyls, clouds, and cacti. I’m leaving Triangle.jack in the directory as a reference to the most basic form of this left scrolling sprite, and will pretty much copy and extend it for the others (though I will put their draw functions in Graphics.jack). I learned a better way to do elif clauses in jack, and a proper way of resetting the triangle position, once it has hit the end of the screen (both of these findings are evident in the file). Also, you can open two instances of the bitmap editor to more seamlessly do the procedure of generating sequences of bitmaps, each 2 pixels shifted. Now that I think about it, it could be useful to write a script to do this for me, so that I only have to draw the bitmap once, and the script will output all 8 shifts of it.

… 2 hours later, and I have not worked on the dino game at all, but I have been working on a tool to help me (and others) create the animation frames for it. I’m simply extending the Sokoban Bitmap Editor to have a large canvas that is entirely optional to fill, with some features such as being able to shift all the pixels in a certain direction.

My current thought about the ground is to have a sequence of 16 (or some other amount) bitmaps that each ground “chunk” will progress through, so that a ground object does not have to change position, but it will still look like movement.

Things I learned

* The Jack platform uses keycodes 131 and 133 for up and down arrow respectively, as opposed to regular ASCII.
* Rather than switch the screen color, draw an object, then switch back, just add eraseObject functions that access the same screen areas as the regular drawObject functions, but poke 0.
* 8 frames per chunk is good in terms of efficiency and smoothness. It is tiring to create all 8 frames, but hopefully this will not be the case for future generations 😊 (if I get my tool extension to work).

Day 4

Well, it’s been almost a week since I got side-tracked with the bitmap editor, but not only is it finished, but it also has some pretty cool features that I had no idea from the start that I’d include. Now that I can mostly automate the process of creating 8 shifted frames of a sprite, I can do just that for the cactus and pterodactyl.

After creating the sprites for the Pterodactyl, I found a bug in the editor. So now I have to fix it.

Day 5

Fixed the bug, but ‘Fit to Drawing’ mode still does not seem to work completely; either that or I do not know how to use it, which would be bad since I created it LOL. Rectangle mode is good though; I should have just stuck to using that mode for the cacti. Now I am stuck with sprites that do not completely erase themselves as they change positions.

Rather than just have the sprite disappear when it gets to “destroy location,” I thought I’d create a terminating sequence of frames in which the cactus moves off the canvas (and so should move off the screen). But this is just something else that I need to debug. I’m going to ignore the problems for now and just create the clouds and try to get the game points and collision to work. For the cloud, I’m just going to use rectangle mode and hope it works.

It did not work. I quit…for now.

Day 5

But I can’t quit. I just do not understand why my bitmap editor is not adding enough buffer on one of the frames. AHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG. Ok, just had to say that. It turns out that the problem was using a ceil instead of floor for negative numbers…which seems like an honest mistake, but it’s not because I had already thought of using floor (which is what I was supposed to), but this meant that GetColumns was including an extra column in many cases that it was not necessary, and this was fine for fit to drawing, but added useless code for rectangle and full canvas. If I was the type to smash something, I’d probably do it, but actually, I’m not really upset because I think this means that the bug was not super hard to find; whereas it could have taken hours, I actually found it (this better be it) in a few minutes.

I just remade my cloud sprite after fixing such bug and it does indeed work 😊. SO HAPPY. Now I just have to go back and remake my cacti too; then I can actually start making the “game” part.

Day 6

I realized that it is better to create an end sequence by just erasing two columns at a time, since this way, the edge word of the screen stays as the edge of the drawing. The only difficulty is that if the top left of the sprite is at all higher than top right, and the editor is set to have top left as base address, then the sprite moves upward during this disappearance. I’m hoping my game is challenging enough that the player does not pay attention to the “graphical weirdness” that will be going on in the clouds. I also have my pterodactyl simply disappear as its end frame, so that’s kinda weird too, but I do not really feel like re-drawing it and creating an end sequence for it. Now I’m going to do the cacti in the same way that I did the cloud, but I set the editor to use bottom left as base address for the end sequence, so that the cactus does not start levitating as it exits the screen.

I actually forgot to switch the base address mode, but I simply added the correct multiple of 32 to memAddress at the beginning of the functions near the end of the cactus, so that it would not levitate. Right now, I only have Cactus2 done, but I’m gonna start working on collisions and maybe pseudorandom positions and then later add Cactus1 again.

For now, I will treat each sprite collision box as a rectangle, and each class will have a method called collision, which will take the coordinates of the bottom right and top left corners of the Trex head and body, and simply ensure that the corner is not within the collision box. This might be somewhat oversimplified, but I hope it will work anyways.

I can’t seem to get an intersection detection method working. I’ll try again on Monday.

Day 7

I was trying to think of a way to get random numbers working, and I think I have an idea to get the initial seed somewhat randomly: I will have users click spacebar to start playing, and count the number of clock cycles that they are holding the spacebar down, and that will be the seed for the random number class.

Day 8 (quite a while later)

Now I need to get collisions to work before anything else can advance.

So it turns out, the rectangle intersection formula I was using was interpreting x and y coordinates as though on an x-y plane, so the comparison of y values was opposite of what it is supposed to be. Now I got collision working, I just gotta set it up for the different sprites.

I got a little side-tracked, trying to fix the Trex jump speed, as it was so fast that it was difficult to even get over a single cactus. It took a while of tweaking, but I finally got a decent combination of gravity rate, initial jump velocity, and percent of clock cycles to actually update gravity and jump position, but now the Trex jumps pretty similarly to how it does in the actual game.

Now, I’m trying to get the Pterodactyl collision box to work but it is being very frustrating since I had been defining height as the number of words above the y = 255 – 22 line (ground level), so all the numbers are weird. I think I might change all of it to be in regular screen coordinates.

Day 9

Ok, I re did part of the Pterodactyl resetting, so that height is now based on screen coordinates, which makes things way nicer. Also, I cached the Trex height, updating it with down-velocity and changing it by + or -16 when entering or exiting from a crouch. This way, it takes less time to compute the collision box.

Yay! This is finally (although very basic atm) the chrome dino game, with jumping, crouching, collisions, and moving obstacles. I can think of two major problems with it that need to be addressed before it realistically resembles the actual game: we need more than just one of each object (and they need to be randomly distributed when reset), we need an actual end screen (rather than just halting the program), and we need a score tracker. The score seems easier, so I will start with it. For now, I will handle score in the same way that it is done in the Pong implementation. I know that the output commands are expensive, but I am somewhat surprised that printing the score each clock cycle slows the game significantly. I’m going to only update the score every few cycles (I would have had to do this anyway, since the score increased way to quickly). The game was still kinda slow, so I changed the delay to 7 (it’s good to know we still have a bit of CPU free, but getting concerned that that 7 will go down so low with the addition of more objects and random number generation that we might not be able to increase the game speed as the user progresses.

Now that the score is done, I will tackle the next easiest thing: and end screen (that will hopefully have an option to restart or quit). I will do this tomorrow, however, since I have run out of my short amount of time today.

Day 10

I set up a simple end screen which simply adds text prompting the user to either press space for restart or q for quit. While I was at it, I started keeping track of high score (during a given game session), and return that from DinoGame.run() and pass it into DinoGame.new(). While setting these up, I came across an interesting bug (that would possibly make a good easter egg) in which I could keep holding down a key and so the end screen passed in a moment and since I was updating score with highscore when creating a DinoGame, the score continued progressing even while crashing. It would be interesting to get this to work using a certain key on the keyboard, but it would require altering and coordinating main, DinoGame, and Endscreen, so it is more complicated than it’s probably worth. I do want to add an easter egg, but I’ll try to find a better idea. Next, I need to get multiple objects, pseudo randomness, and combine those to get a random distribution of obstacles on the map.

I started to write my own linear congruential generator, but then I realized that it was the same algorithm as used in the FlappyBird random, so I just copied it, simply to save time tweaking the initial constants.

The randomness works kinda ok, but there are too often clusters that are actually impossible to jump. I’ve increased the jump velocity a little, which helps, but it can still be frustrating. So thought I could compile a list of “good” seeds, and just randomly select one at the beginning.

Good seeds: 12894, 31192

YAY! Finally, Dino 1.0 is complete!!!! It has taken long enough! The are some glitchy graphics still, and areas that I could overall improve the game, which I will do for version 1.1, but for now, it’s done. I’m also planning some ‘glitches’/’features’/’easter eggs’ to add in a separate release, since my version of the game is really not as fun as the original by Google, since my obstacle distribution is weird, and there is no speed or game progression (yet…).

Deluxe version:

Will have various achievements to go along with the easter eggs. Using the code that enables crouching at the peak of a jump (must have jump velocity as a negative multiple of gravity), there will be an achievement, “Who knew dinosaurs could fly?” (in this mode, the user could go on indefinitely, so I think we should add an extra clock counter to Trex and have it reset to standing every 1000 cycles or so). There will be a secret menu (perhaps accessed by clicking the ‘?’ mark) that will tell the user about various keys that do cool things. For example, when starting the game, pressing a number key could determine the jumping scale factor (or could use right/left arrow keys to change gravity). Maybe I could add new Trex ‘skins,’ or add a dark mode (though that would take a lot of effort in creating all the bitmaps again). There should also be a few score changing achievements/features. The github readme should showcase the features in a fun way that does not give away how to actually do them.

Day 11