Dinosaur

RetroLCD.com

Contents

RETRoLCD.com

About Dinosaur

To start playing Dinosaur in the Chrome Web Browser put

chrome://dino

In the URL bar

← → C   Chrome | chrome://dino
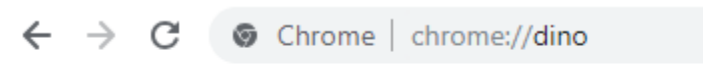
Initially you will see

No internet

Try:
- Checking the network cables, modem, and router
- Reconnecting to Wi-Fi

ERR_INTERNET_DISCONNECTED

Press the up arrow to start playing

Press the up arrow to jump and avoid the cacti

When you hit a cactus, you see

Breaking It Down

The first thing we want to consider is what the controls are. In this version of the game, there is only one button: up. You can jump. That's it.

You may have noticed already that the controller for the Arduino version uses 4 buttons. We could have gotten away with one, but rather than simply make an exact copy of the game, it will be enhanced a bit.

Embrace, Extend, Extinguish

In other words, apply your own creativity to a base idea so that at some point, your version is completely unrecognizable from the source material.

Now that we know what the controls are, we look at the graphics. It's probably best to think of the graphics in terms of the hero and the enemies.

Our hero is the dinosaur. He's the character the user controls.

Our enemies are the cacti. They are to be avoided. There are two types: big and small.

Now, we need to consider how they move.

Our hero jumps.

Our cacti run.

When breaking things down it's important to look at patterns. We can think of the game in terms of the hero moving or in terms of the cacti moving. And in this case, it's easier to think of it in terms of the cacti running towards the dinosaur.

You could imagine that they are cars and we have some crazy person leaping over them rather than get hit by them as they pass.

Next, we think about how the player scores points. Some games have an ending. This game does not. You simply get one point for each cactus you jump over.

There are no hit points. As soon as you hit a cactus, the game is over.

Embrace

Step 1 is to embrace the game as it is presented.

We'll need a controller with at least 1 button.

We'll need to design graphics for a dinosaur and two cacti.

Use paper to draw out your version of a dinosaur, a large cactus and a small cactus.
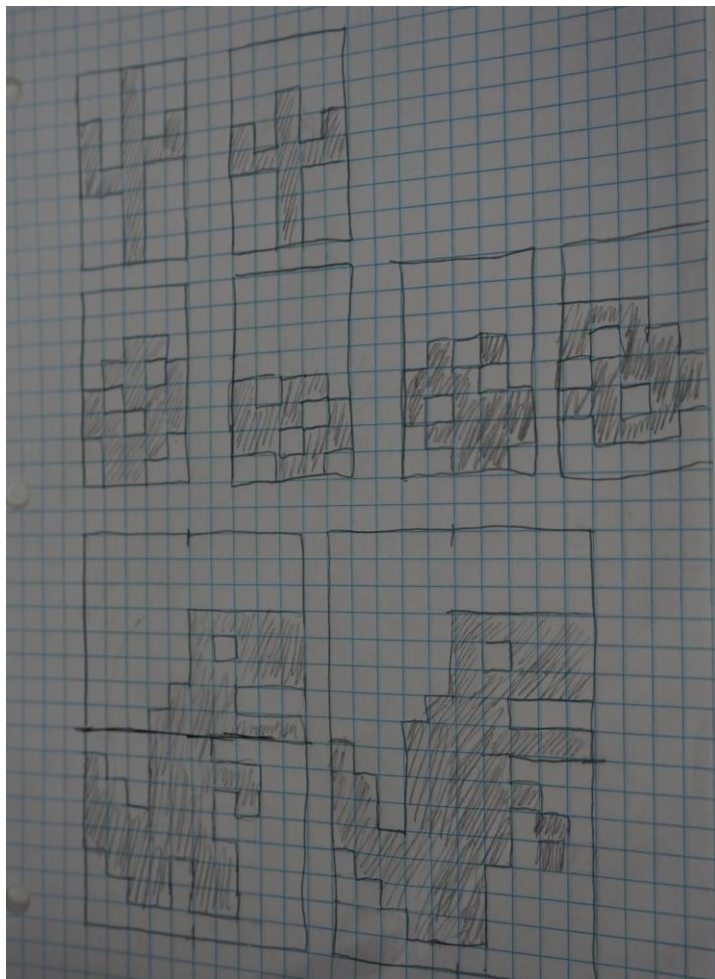
It doesn't need to be fancy. The thing to keep in mind is the limitations of the graphics display we'll be using. We'll be using a 16x2 character display which limits us to 5x8 pixels for each sprite. However, we can combine sprites to make bigger sprites. The other limitation to consider is that we can only have 7 custom sprites loaded into the display at a time. We can swap them out, but only 7 at a time.

The way I've designed my sprites is that the cacti use one sprite each, the tumbleweed has 4 frames of animation and the hero takes up 4 sprites at once and has 4 animation frames.

That gives us 2 cacti sprites, 1 tumble weed sprite and 4 hero sprites that will be loaded onto the display at any given time which maxes out the available 7.

Extend

You probably noticed that there is no tumbleweed in the original game. This is the first extension. Rather than only having the hero jump over enemies, there is a new enemy: the tumbleweed which the hero must dodge by either jumping over it while it's on the ground, or duck under it while it's in the air.

The tumbleweed bounces as it approaches the player using a simple SIN function
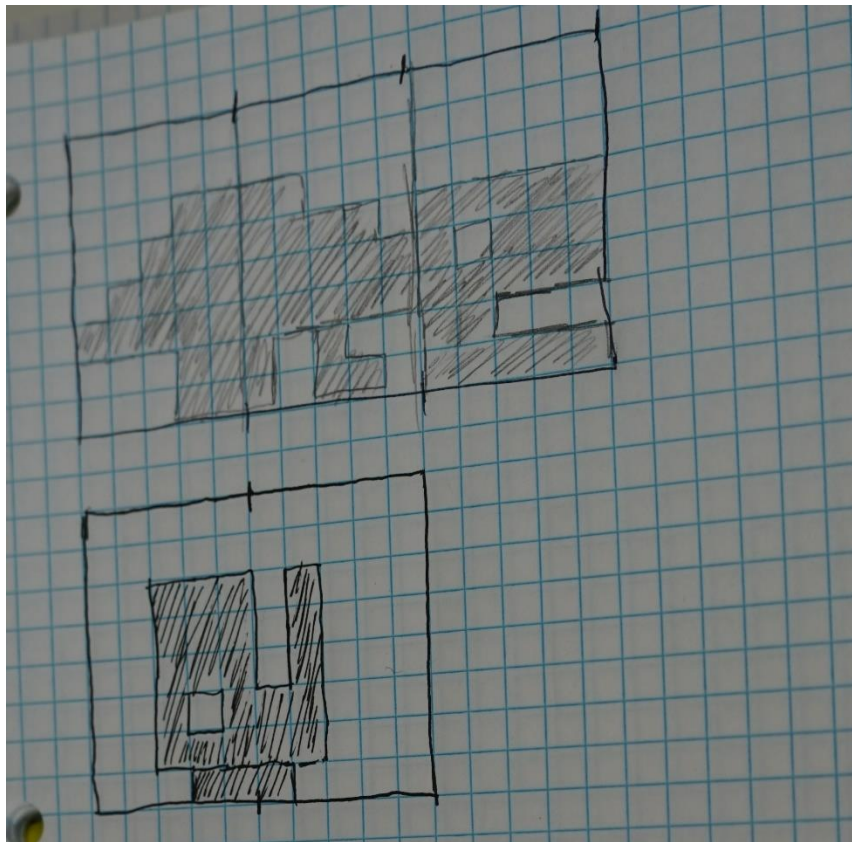
Since our hero can jump, we need a jump animation frame. As our screen space is limited, the hero goes into a crouch position on the first or second line only of the display depending on whether he's jumping or ducking.

When our hero is hit, we don't need to change the body, so we just have two additional sprites that make up his head when he throws his head back in pain from getting hit.

Being able to duck is the second extension.

And finally, rather than have a game over from getting hit, the score will just decrease until it reaches zero. If players want to compete, they can simply set a time limit and see who can get the highest score in a fixed amount of time.

Extinguish

How would you change the controls, graphics, scoring and "plot" of the game while still being limited to 7 different sprites on the screen at a time and a 16x2 display?

Coding the Game

When it comes to coding any game there is a good set of steps to help you break down the problem so you can tackle pieces at a time in a logical progression.

1. Set Up the Screen
2. Draw a pixel
3. Move a pixel
4. Draw more pixels
5. Move pixels based on user input
6. Recognize and react to collisions between pixels
7. Implement scoring rules

Set Up the Screen

Dinosaur makes use of the LiquidCrystal_I2C library which allows us to talk to the character display over I2C which requires far fewer wires than trying to talk directly to its data lines.

```
2 #include "LiquidCrystal_I2C.h"
```

This library is included with the downloadable source code to avoid issues with updates or variations of the library that don't work with the rest of the code.

Next, we configure the library for the specific display we are using.

```
 8 // https://forum.arduino.cc/index.php?topic=117045.0
 9 // Set the LCD address to 0x27 for a 16 chars and 2 line display
10 LiquidCrystal_I2C lcd(0x3F, 16, 2);
```

Namely, the address and resolution. We are using a 16x2 character display and it is configured to be at address 0x3F. Notice that the comment says 0x27. This is because 0x27 is often the address specified in sample code as it is often the address set. But not always.

The location of the character display is set by the factory. There are three pairs of connectors on the board that can be connected giving 7 additional address lines that can be selected.

Note the three sets of connectors above the blue square on the left. They can be soldered together to change the address of the controller.

https://retrolcd.com/Help/I2CFinder

Use this tool to figure out what the memory location of your display is. Make sure your display is properly connected to the Arduino before running it. It cycles through all the possible addresses and indicates which ones have a device connected. Don't connect more than one unknown device at a time or you won't know which address is for which device. Fortunately, for this project, we only need one.

```
39    // initialize the LCD
40    lcd.begin();
41    lcd.clear();
42    lcd.backlight();
43    lcd.setCursor(0, 0);
44
```

There are three commands really needed in the Setup function:

1. Begin
2. Clear
3. Backlight

Begin tells the controller to start listening. Clear, clears out the display so it is empty of any characters. And Backlight turns on the backlight. The blue square on the controller is what adjusts the brightness / contrast of the display.

```
105      lcd.setCursor(0, y);
106      lcd.print(line);
```

The print function works just like Serial.print and prints out whatever is contained in the string.

Draw a Pixel

When it comes to character displays, we don't really have pixels. It's generally not recommended to try to force something to do something it wasn't designed to do.

Since we are working with a character display, the first task is to put a character on the display in a user-defined location.

This shows that you have control of the display.

If you can put the letter "A" on the character display at a specific location, you have complete control over it.

Whenever you are working with graphics it is important to have a double buffer. The first buffer is visible to the user. This is the character display. The second buffer is where we build up what will be shown on the first buffer. The character display does not provide a second buffer and so we must implement our own.

```
12 char Screen[32];
13 int screen_x = 16;
14 int screen_y = 2;
```

Since we are working with printed text, the buffer needs to be an array of char data types. Notice that the screen array size is equal to screen_x * screen_y. We could have defined Screen as [2][16] but it is easier to just use a single dimension array.

Rather than draw directly to the LCD Display, we will now write a few functions to draw to our second buffer so that the user isn't subjected to visual artifacts like flickering as we clear the screen and redraw it.

```
62 void ClearScreen() {
63   for (int j = 0; j < 32; j++) {
64     Screen[j] = ' ';
65   }
66 }
```

The first thing we need is a way to clear out the buffer so it's ready for the next frame. There are two things to notice here: the first is that we are not referencing screen_x or screen_y, we are just using the fixed value of 32. This saves us a calculation as we're not writing a generic library anyway. If you wanted these functions to be compatible with a variety of displays, it would be necessary to use the variables rather than a fixed value.

The second thing to notice is that we're not writing zero to each of the array elements. Instead we are using the space character. This is because zero means it is the end of the string. Remember, we are using a character display and will be writing strings to it, so we must follow the rules of strings. And the most fundamental rule of strings is that they must end in a null (aka zero) value.

The next function we need is the PlotCHAR function which will place characters into our buffer at a specified location.

```
67  void PlotCHAR(int x, int y, char c) {
68    if (x + y * screen_x >= 32) {
69      return;
70    }
71    if (x < 0) {
72      return;
73    }
74    if (y < 0) {
75      return;
76    }
77    if (x > 15) {
78      return;
79    }
80    if (y > 1) {
81      return;
82    }
83    Screen[x + y * screen_x] = c;
84  }
```

The first check is there to demonstrate how to ensure that we are not writing outside the bounds of the buffer. However, it does not check that the given x and y values are valid for our physical display. For example, if we passed in y = 0 and x = 24, that would not extend past the boundary of the buffer, but our display is only 16 characters wide so the character would end up on the 2nd line which is not the expected behavior.

The four conditions after the first ensure that the given x and y values are within the bounds of the physical display.   If they aren't, the function returns and does not put the given CHAR in the buffer.

If x and y fit in our array and they are within the physical bounds of the display, they we use the simple math function to store the character in the buffer.
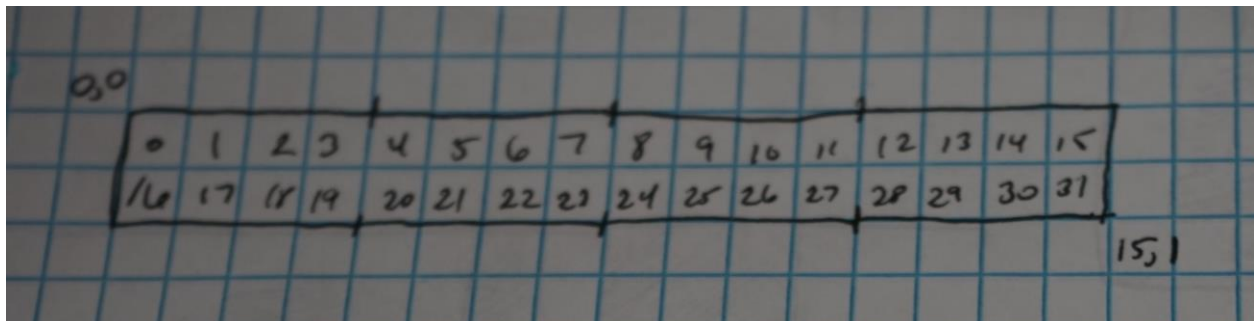
When designing these systems, it is best to draw things out. Especially when they're small. Once you know the rule, it scales to any size. You can see from the picture that the second half of the array corresponds to the second row of the display. Multiplying y by the width of the physical display gives us the section of the buffer that corresponds to the physical row.

Notice that we are using the CHAR variable type. Since we are dealing with printed characters, we need to use a signed variable, or the display will not interpret the characters correctly.

And of course, we need a way to push this virtual screen to the physical screen.

```
88    char line[17];

100   for (y = 0; y < screen_y; y++) {
101     for (x = 0; x < screen_x; x++) {
102       line[x] = Screen[x + y * screen_x];
103     }
104     line[16] = 0;
105     lcd.setCursor(0, y);
106     lcd.print(line);
107   }
```

Notice that our line variable has 17 characters. This is because we need to put in a character zero or the display will not stop trying to read characters after the first 16.

What we are doing here is copying 16 bytes of our buffer at a time and printing them out to the screen.

There is a memcpy function which could replace the x loop, but this is our first game project and we'll stick to a more basic solution.

By writing whole lines at a time, we avoid the problem of flicker had we just cleared and drawn directly on the physical display. Using spaces allows us to clear our screen and put out the new display in one step.

We could reduce this to writing a single line to the display using a carriage return, memcpy and by increasing the line array. Or even by modifying our main screen buffer and how it calculates where to put characters in it. I will leave that as a challenge to readers.

Now we know how to define a custom buffer for a display. Write to it. And push it out to the physical display.

Move a Pixel

Now that we can plot to our screen, it's time to learn how to move a pixel. For this step we will think about our game and how our character moves. In the Chrome version, the dinosaur only moves up when it jumps. So, we could say, "we just need a y position." But we're working with a screen that only has 2 y positions and we're introducing a tumbleweed that bounces so our character may need to move along the x axis in order to position themselves to be able to avoid being hit. It is common in auto-scrolling games that the player has some freedom of movement in the same axis the game is scrolling.

Jumping is going to be handled a bit differently which we will cover later. So, for our movement we will stick to moving left and right.

```
19 int player_x;
```

That gives us a single variable that we need to represent the position of the player.

Then, in our setup function we have

```
48    player_x = 0;
```

In our loop function we can change player_x and use the PlotCHAR function to see the character we're plotting move on the display.

I'm not going to put the code for this here and will leave it as an exercise for the reader to create working code that moves a character back and forth on the display.

Draw More Pixels

Before we start drawing more pixels on the screen we need to think about their purpose and how they will move. We will be having our cacti move towards the player so the player position will not be used to calculate the position of the cacti. They are all separate entities. And we want to be able to move the cacti and some controllable speed. There are several ways to do this. One way is to move the cacti only certain frame numbers and have a fixed frame rate. The other is to use the float variable type so we can adjust their position in very small increments and then round when rendering their position.

```
30 float cacti_x[10];
```

Again, everything only really moves along the x-axis. This gives us 10 cacti that can be going at once. When a cactus reaches the left side of the screen it will reset to another random x position.

We only need an array big enough to hold as many cacti as can be displayed on the screen at once.

```
56    for (j = 0; j < cacti_count; j++) {
57      cacti_x[j] = random(17, max_x);
58      cacti_sprite[j] = random(6, 8);
59    }
```

In our setup function we can now loop through the array of cacti and set their initial x positions. We will cover what cacti_sprite is later.

The variable max_x is a defined as a global and is set to 500. That allows our cacti to be up to 500 characters away from the left side of the screen. This is how it change the distance between cacti. Because the minimum of the random function is 17, they will never immediately appear on the screen. And by having a maximum of 500, it can give the player a little break before they reach the visible screen.

In our loop function we now add

```
260   int j;
261   for (j = 0; j < cacti_count; j++) {
262     PlotCHAR((int)cacti_x[j], 1, cacti_sprite[j]);
263   }
```

Notice that we cast the cacti_x variable to an int which cuts off the decimal portion. And they are always rendered at y position 1 which is the bottom row of the screen.

Remember we also added a tumbleweed.

```
23 float weed_x;
24 float weed_y;
```

The tumbleweed can be in the top row or bottom row, so we need a y value as well as an x value.

In the setup function, we default the x location of the weed to be

```
54   weed_x = random(200, 1000);
```

Which puts the weed well off screen, so the player does not encounter it right away.

```
165   weed_y = (int)(sin(total_time * 6.0 / 3.14) + 1);
```

To calculate the y position of the weed we use the sin function along with the total time which goes from 0 to 60 seconds.

In our loop function we now have

```
240   PlotCHAR((int) weed_x, (int) weed_y, 5);
```

Which renders the weed based on the rounded x and y values. We'll cover what the "5" is later.

Our cacti move using a simple bit of math

```
167   int j;
168   for (j = 0; j < cacti_count; j++) {
169     cacti_x[j] -= frame_time * 4.0;
```

This causes them to move 4 spaces per second.

Move pixels based on user input

We'll be using the controller library found here

https://retrolcd.com/KeyInCode

Type it in once, and then you can reuse it as many times as you want.

```
6 #include "Controller.h"
```

In Controller.h you will find four #define macros

```
 6 // I/O Pins used by controller - 4 Required
 7 #define CONTROLLER_BUTTON_PIN_A 3
 8 #define CONTROLLER_BUTTON_PIN_B 4
 9 #define CONTROLLER_BUTTON_PIN_C 5
10
11 #define CONTROLLER_BUTTON_PIN_READ 7
```

Make sure you either wire up the controller to the same pins or update Controller.h to match your wiring.

Then in our loop function we have

```
231    Controller::ReadButtons();
```

And call our HandleInput function

```
110 void HandleInput()
111 {
112   if (dino.dino_state != DINO_STATE_JUMP
113       && dino.dino_state != DINO_STATE_CROUCH) {
114     if (Controller::IsPressedAgain(CONTROLLER_BUTTON_A)) {
115       dino.Jump();
116       Controller::MarkUnreleased(CONTROLLER_BUTTON_A);
117     } else {
118       if (Controller::IsPressedAgain(CONTROLLER_BUTTON_B)) {
119         dino.Crouch();
120         Controller::MarkUnreleased(CONTROLLER_BUTTON_B);
121       } else {
122         if (Controller::IsPressedAgain(CONTROLLER_BUTTON_LEFT)) {
123           if (player_x > 0) {
124             player_x--;
125           }
126           Controller::MarkUnreleased(CONTROLLER_BUTTON_LEFT);
127         } else {
128           if (Controller::IsPressedAgain(CONTROLLER_BUTTON_RIGHT)) {
129             if (player_x < 4) {
130               player_x++;
131             }
132             Controller::MarkUnreleased(CONTROLLER_BUTTON_RIGHT);
133           }
134         }
135       }
136     }
137   }
138 }
```

We are using the IsPressedAgain and MarkUnreleased methods so that the user cannot just hold down the button. They must press and then release the button before it can be pressed again.

There is very simple logic for the controllers. The user and move the hero left or right and they can jump or crouch.

We'll ignore the dino class for now. All we are doing is checking to see if buttons are pressed and modifying variables depending on which buttons are pressed. Notice that we are using a series of if-else statements. This prevents the user from pressing more than one button per frame. It is especially important for the Jump and Crouch routines as they are states and starting both would cause problems. Once the player is in the Jump or Crouch state they cannot go into another state until those states complete. The first "if" verifies that the user is not in a jump or crouch state before processing any input.

Recognize and react to collisions between pixels

Now that everything is moving around the screen it's time to figure out when things hit each other. Or when things go out of bound.

The rules of the game are that when something reaches the left side of the screen without hitting the player, the player gets points. If something hits the player, the position of it resets and the player loses points.

The logic which handles objects reaching the left side of the screen is found in the function HandleSpriteUpdates

```
158 void HandleSpriteUpdates()
159 {
160   weed_x -= frame_time * 4.0;
161   if (weed_x < 0) {
162     points += 10;
163     weed_x = random(200, 1000);
164   }
165   weed_y = (int)(sin(total_time * 6.0 / 3.14) + 1);
166
167   int j;
168   for (j = 0; j < cacti_count; j++) {
169     cacti_x[j] -= frame_time * 4.0;
170     if (cacti_x[j] < 0) {
171       points++;
172       cacti_x[j] = random(17, max_x);
173       cacti_sprite[j] = random(6, 8);
174     }
175   }
176 }
```

This is where we move the cacti and the weed left every frame and, also check to see if it has gone off the left side of the screen. If a cactus goes off the left side of the screen, then the player gets a point. If the weed goes off the left side of the screen, the player gets 10 points.

In either case, the x location for the enemy resets.

In the case of the cactus, the sprite can also change.

On the Arduino, the minimum value of random is inclusive but the upper bound is exclusive. This means that although the random function is passed 6 and 8, it can only return 6 or 7.

We'll cover sprites later.

In addition to the boundary detection, we also need to detect whether the cacti or weed have hit the player. For that we have a dedicated function to help keep the code manageable.

```
178  void HandleCollision()
179  {
180    switch (dino.dino_state) {
181      case DINO_STATE_JUMP:
182        if (weed_y == 0)
183        {
184          if (weed_x >= player_x && weed_x <= player_x + 2) {
185            weed_x = random(200, 1000);
186            points -= 5;
187            dino.Hit();
188          }
189        }
190        break;
191      case DINO_STATE_CROUCH:
192        if (weed_y == 1)
193        {
194          if (weed_x >= player_x && weed_x <= player_x + 2) {
195            weed_x = random(200, 1000);
196            points -= 5;
197            dino.Hit();
198          }
199        }
200        break;
201      default:
202        if (weed_x >= player_x && weed_x <= player_x + 1) {
203          weed_x = random(200, 1000);
204          points -= 5;
205          dino.Hit();
206        }
207        break;
208    }
209
210    if (dino.dino_state != DINO_STATE_JUMP) {
211      int j;
212      for (j = 0; j < cacti_count; j++) {
213        if (player_x == (int)cacti_x[j]) {
214          cacti_x[j] = random(17, max_x);
215          cacti_sprite[j] = random(6, 8);
216          points--;
217          dino.Hit();
218          break;
219        }
220
221      }
222    }
223    if (points < 0) {
224      points = 0;
225    }
226  }
```

We start off by checking to see if the player has collided with the weed. The weed can be up in the air or down on the ground. The player can be standing, jumping or crouching.   It may help to create a truth table to ensure all the possibilities are covered.

Player Is	Weed Is	Collision
Standing	In Air	Yes
Standing	On Ground	Yes
Crouching	In Air	No
Crouching	On Ground	Yes
Jumping	In Air	Yes
Jumping	On Ground	No

We can see from this truth table that there is a 66% chance that the weed is going to hit the player.

Therefore, we give 10 points to the player for dodging it and only take away 5 points if they get hit. In 6 attempts they will gain 10 points twice and lose 5 points 4 times. Which works out to zero points given completely random chance. That leaves skill as the deciding factor.

If the dino is in the jump state, then we check to see if the weed is Up. If it's not, there's no chance of collision. If it is, then we check to see if the weed's x position is within the player sprite. And if so, there is a collision, points are lost and the weed resets.

If the dino is in the crouch state, then we check to see if the weed is Down. If it's not, there's no chance of collision. If it is, then we check to see if the weed's x position is within the player sprite. And if so, there is a collision, points are lost and the weed resets.

If the dino is in the default state, which is standing, then we only need to check to see if the weed's x position is within the player's sprite and if so, points are lost and the weed resets.

When it comes to the cacti, we just check to see if the dino is jumping, and if it's not, we see if the cactus is in the same position as the player. Notice that in the case of the cactus, there is only one space being checked while with the weed, three spaces are being checked when jumping or crouching and two are being checked when standing. This is because the jump and crouch sprite are 3 characters wide and the standing sprite is 2 characters wide. For the cactus, we're just making it a little easier for the player to avoid them. There is a bit of extra time to jump out of the way.

Dinosaur.ino

```cpp
1  #include <Wire.h>
2  #include "LiquidCrystal_I2C.h"
3  #include "DinoController.h"
4  #include "WeedController.h"
5  #include "CactiController.h"
6  #include "Controller.h"
7
8  // https://forum.arduino.cc/index.php?topic=117045.0
9  // Set the LCD address to 0x27 for a 16 chars and 2 line display
10 LiquidCrystal_I2C lcd(0x3F, 16, 2);
11
12 char Screen[32];
13 int screen_x = 16;
14 int screen_y = 2;
15 int start_timer;
16 int end_timer;
17 float frame_time;
18 float total_time;
19 int player_x;
20 int max_x = 500;
21 int points = 0;
22
23 float weed_x;
24 float weed_y;
25
26 DinoController dino;
27 WeedController weed;
28 CactiController cacti;
29
30 float cacti_x[10];
31 int cacti_sprite[10];
32 int cacti_count = 10;
```

```
34  void setup() {
35    Serial.begin(9600);
36
37    randomSeed(analogRead(0));
38
39    // initialize the LCD
40    lcd.begin();
41    lcd.clear();
42    lcd.backlight();
43    lcd.setCursor(0, 0);
44
45    dino.Init();
46    weed.Init();
47    Controller::Init();
48    player_x = 0;
49
50    lcd.createChar(6, cacti.frame1);
51    lcd.createChar(7, cacti.frame2);
52
53    total_time = 0;
54    weed_x = random(200, 1000);
55    int j;
56    for (j = 0; j < cacti_count; j++) {
57      cacti_x[j] = random(17, max_x);
58      cacti_sprite[j] = random(6, 8);
59    }
60  }

62  void ClearScreen() {
63    for (int j = 0; j < 32; j++) {
64      Screen[j] = ' ';
65    }
66  }
```

```
68  void PlotCHAR(int x, int y, char c) {
69    if (x + y * screen_x >= 32) {
70      return;
71    }
72    if (x < 0) {
73      return;
74    }
75    if (y < 0) {
76      return;
77    }
78    if (x > 15) {
79      return;
80    }
81    if (y > 1) {
82      return;
83    }
84    Screen[x + y * screen_x] = c;
85  }

87  void RenderScreen() {
88    int x, y;
89    char line[17];
90    byte point_char[16];
91
92    String point_disp = "Pts:" + String(points);
93    point_disp.getBytes(point_char, 16);
94
95    for (x = 0; x < 16; x++) {
96      PlotCHAR(x + 6, 0, (char)point_char[x]);
97    }
98
99    lcd.setCursor(0, 0);
100
101   for (y = 0; y < screen_y; y++) {
102     for (x = 0; x < screen_x; x++) {
103       line[x] = Screen[x + y * screen_x];
104     }
105     line[16] = 0;
106     lcd.setCursor(0, y);
107     lcd.print(line);
108   }
109 }
```

```
111  void HandleInput()
112  {
113    if (dino.dino_state != DINO_STATE_JUMP
114        && dino.dino_state != DINO_STATE_CROUCH) {
115      if (Controller::IsPressedAgain(CONTROLLER_BUTTON_A)) {
116        dino.Jump();
117        Controller::MarkUnreleased(CONTROLLER_BUTTON_A);
118      } else {
119        if (Controller::IsPressedAgain(CONTROLLER_BUTTON_B)) {
120          dino.Crouch();
121          Controller::MarkUnreleased(CONTROLLER_BUTTON_B);
122        } else {
123          if (Controller::IsPressedAgain(CONTROLLER_BUTTON_LEFT)) {
124            if (player_x > 0) {
125              player_x--;
126            }
127            Controller::MarkUnreleased(CONTROLLER_BUTTON_LEFT);
128          } else {
129            if (Controller::IsPressedAgain(CONTROLLER_BUTTON_RIGHT)) {
130              if (player_x < 4) {
131                player_x++;
132              }
133              Controller::MarkUnreleased(CONTROLLER_BUTTON_RIGHT);
134            }
135          }
136        }
137      }
138    }
139  }

141  void HandleSpriteChange()
142  {
143    if (weed.weed_sprite_change) {
144      lcd.createChar(5, weed.frame);
145      weed.weed_sprite_change = false;
146    }
147
148    if (dino.dino_sprite_change) {
149      // NOTE: do not use createChar(0, ...), it confuses the Ardino
150      // http://forum.arduino.cc/index.php?topic=74666.0
151      lcd.createChar(1, dino.dinoTL);
152      lcd.createChar(2, dino.dinoTR);
153      lcd.createChar(3, dino.dinoBL);
154      lcd.createChar(4, dino.dinoBR);
155      dino.dino_sprite_change = false;
156    }
157  }
```

```
159  void HandleSpriteUpdates()
160  {
161    weed_x -= frame_time * 4.0;
162    if (weed_x < 0) {
163      points += 10;
164      weed_x = random(200, 1000);
165    }
166    weed_y = (int)(sin(total_time * 6.0 / 3.14) + 1);
167
168    int j;
169    for (j = 0; j < cacti_count; j++) {
170      cacti_x[j] -= frame_time * 4.0;
171      if (cacti_x[j] < 0) {
172        points++;
173        cacti_x[j] = random(17, max_x);
174        cacti_sprite[j] = random(6, 8);
175      }
176    }
177  }
```

```
179  void HandleCollision()
180  {
181    switch (dino.dino_state) {
182      case DINO_STATE_JUMP:
183        if (weed_y == 0)
184        {
185          if (weed_x >= player_x && weed_x <= player_x + 2) {
186            weed_x = random(200, 1000);
187            points -= 5;
188            dino.Hit();
189          }
190        }
191        break;
192      case DINO_STATE_CROUCH:
193        if (weed_y == 1)
194        {
195          if (weed_x >= player_x && weed_x <= player_x + 2) {
196            weed_x = random(200, 1000);
197            points -= 5;
198            dino.Hit();
199          }
200        }
201        break;
202      default:
203        if (weed_x >= player_x && weed_x <= player_x + 1) {
204          weed_x = random(200, 1000);
205          points -= 5;
206          dino.Hit();
207        }
208        break;
209    }
210
211    if (dino.dino_state != DINO_STATE_JUMP) {
212      int j;
213      for (j = 0; j < cacti_count; j++) {
214        if (player_x == (int)cacti_x[j]) {
215          cacti_x[j] = random(17, max_x);
216          cacti_sprite[j] = random(6, 8);
217          points--;
218          dino.Hit();
219          break;
220        }
221
222      }
223    }
224    if (points < 0) {
225      points = 0;
226    }
227  }
```

```
229  void loop() {
230    start_timer = millis();
231
232    Controller::ReadButtons();
233
234    HandleInput();
235    HandleSpriteChange();
236    HandleSpriteUpdates();
237    HandleCollision();
238
239
240    ClearScreen();
241    PlotCHAR((int) weed_x, (int) weed_y, 5);
242    switch (dino.dino_state) {
243      case DINO_STATE_JUMP:
244        PlotCHAR(player_x + 0, 0, 1);
245        PlotCHAR(player_x + 1, 0, 2);
246        PlotCHAR(player_x + 2, 0, 3);
247        break;
248      case DINO_STATE_CROUCH:
249        PlotCHAR(player_x + 0, 1, 1);
250        PlotCHAR(player_x + 1, 1, 2);
251        PlotCHAR(player_x + 2, 1, 3);
252        break;
253      default:
254        PlotCHAR(player_x + 0, 0, 1);
255        PlotCHAR(player_x + 1, 0, 2);
256        PlotCHAR(player_x + 0, 1, 3);
257        PlotCHAR(player_x + 1, 1, 4);
258        break;
259    }
260
261    int j;
262    for (j = 0; j < cacti_count; j++) {
263      PlotCHAR((int)cacti_x[j], 1, cacti_sprite[j]);
264    }
265
266    RenderScreen();
267
268    end_timer = millis();
269    frame_time = (float) (end_timer - start_timer) / 1000.0;
270    dino.Update(frame_time);
271    weed.Update(frame_time);
272
273    total_time += frame_time;
274    total_time = ((total_time / 60.0) - (int) (total_time / 60.0)) * 60.0;
275
276  }
```

CactiController.h

```
1  #ifndef CactiController_h
2  #define CactiController_h
3
4  #include "Arduino.h"
5  #include "LiquidCrystal_I2C.h"
6
7  class CactiController {
8    public:
9      static byte frame1[8] = {
10       0b00100,
11       0b00100,
12       0b10101,
13       0b10111,
14       0b11100,
15       0b00100,
16       0b00100,
17       0b00100
18     };
19
20     static byte frame2[8] = {
21       0b00000,
22       0b00000,
23       0b00100,
24       0b10101,
25       0b11111,
26       0b00100,
27       0b00100,
28       0b00100
29     };
30 };
31
32 #endif
```

CactiController.cpp

```
1  #include "Arduino.h"
2  #include "CactiController.h"
3
4  // needs to be define here with the actual value in the h file
5  static byte CactiController::frame1[8];
6  static byte CactiController::frame2[8];
7
```

DinoController.h

```
1  #ifndef DinoController_h
2  #define DinoController_h
3
4  #define DINO_STATE_WALK1_TRANSITION 1
5  #define DINO_STATE_WALK1 2
6
7  #define DINO_STATE_WALK2_TRANSITION 3
8  #define DINO_STATE_WALK2 4
9
10 #define DINO_STATE_JUMP 5
11 #define DINO_STATE_CROUCH 6
12
13 #include "Arduino.h"
14 #include "LiquidCrystal_I2C.h"
15
16 class DinoController {
17   private:
18     static byte dinoHitL[8] = {
19       0b00000,
20       0b00000,
21       0b00111,
22       0b00111,
23       0b00111,
24       0b00101,
25       0b00111,
26       0b00011
27     };
28
29     static byte dinoHitR[8] = {
30       0b00000,
31       0b00000,
32       0b01000,
33       0b01000,
34       0b01000,
35       0b01000,
36       0b11000,
37       0b10000
38     };
39
40     static byte dinoTL1[8] = {
41       0b00000,
42       0b00000,
43       0b00000,
44       0b00000,
45       0b00000,
46       0b00000,
47       0b00001,
48       0b00011
49     };
```

```
51    static byte dinoTR1[8] = {
52        0b00000,
53        0b00000,
54        0b00000,
55        0b11111,
56        0b10111,
57        0b11111,
58        0b10000,
59        0b11111
60    };
61
62    static byte dinoBL1[8] = {
63        0b00011,
64        0b10011,
65        0b11011,
66        0b11111,
67        0b11111,
68        0b01110,
69        0b00110,
70        0b00000
71    };
72
73    static byte dinoBR1[8] = {
74        0b10000,
75        0b11100,
76        0b10100,
77        0b10000,
78        0b11000,
79        0b11000,
80        0b11000,
81        0b00000
82    };
83
84    static byte dinoBL2[8] = {
85        0b10011,
86        0b11011,
87        0b11011,
88        0b11111,
89        0b01111,
90        0b00111,
91        0b00011,
92        0b00000
93    };
94
95    static byte dinoBR2[8] = {
96        0b10000,
97        0b11000,
98        0b10100,
99        0b10100,
100       0b10000,
101       0b10000,
102       0b10000,
103       0b00000
104   };
105
106   static byte JumpCrouch1[8] = {
107       0b00000,
108       0b00000,
109       0b00011,
110       0b00111,
111       0b01111,
112       0b11111,
113       0b00011,
114       0b00011
115   };
```

```
117     static byte JumpCrouch2[8] = {
118        0b00000,
119        0b00000,
120        0b11000,
121        0b11110,
122        0b11111,
123        0b11111,
124        0b10100,
125        0b10110
126     };
127
128     static byte JumpCrouch3[8] = {
129        0b00000,
130        0b00000,
131        0b00000,
132        0b11111,
133        0b10111,
134        0b11111,
135        0b11000,
136        0b11111
137     };
138
139     float dino_state_time;
140     float dino_hit_timer;
141
142     float dino_state_duration;
143
144     void NextState();
145     void SetWalk1();
146     void SetWalk2();
147     void SetJumpCrouch();
148
149  public:
150
151     byte dinoTL[8];
152     byte dinoTR[8];
153     byte dinoBL[8];
154     byte dinoBR[8];
155     bool dino_sprite_change;
156     int dino_state;
157     bool dino_is_hit;
158
159     void Init();
160     void Update(float s);
161     void Jump();
162     void Crouch();
163     void Hit();
164
165 };
166
167 #endif
```

DinoController.cpp

```cpp
1  #include "Arduino.h"
2  #include "DinoController.h"
3
4  // needs to be define here with the actual value in the h file
5  static byte DinoController::dinoHitL[8];
6  static byte DinoController::dinoHitR[8];
7
8  static byte DinoController::dinoTL1[8];
9  static byte DinoController::dinoTR1[8];
10 static byte DinoController::dinoBL1[8];
11 static byte DinoController::dinoBR1[8];
12 static byte DinoController::dinoBL2[8];
13 static byte DinoController::dinoBR2[8];
14
15 static byte DinoController::JumpCrouch1[8];
16 static byte DinoController::JumpCrouch2[8];
17 static byte DinoController::JumpCrouch3[8];
18
19 void DinoController::Init() {
20   dino_state = DINO_STATE_WALK1_TRANSITION;
21   dino_state_time = 0;
22   dino_state_duration = 0;
23   dino_is_hit = false;
24   SetWalk1();
25 }
26
27 void DinoController::Hit() {
28   if (dino_is_hit) {
29     return;
30   }
31   dino_state = DINO_STATE_WALK1_TRANSITION;
32   dino_state_time = 0;
33   dino_state_duration = 0;
34   dino_hit_timer = 1.0;
35   dino_is_hit = true;
36   SetWalk1();
37 }
38 void DinoController::Update(float s) {
39   dino_state_time += s;
40   if (dino_state_time >= dino_state_duration) {
41     NextState();
42   }
43   if (dino_is_hit) {
44     dino_hit_timer -= s;
45     if (dino_hit_timer <= 0) {
46       dino_hit_timer = 0;
47       dino_is_hit = false;
48     }
49   }
50 }
```

```
52  void DinoController::SetWalk1() {
53    int j;
54    for (j = 0; j < 8; j++) {
55      if (dino_is_hit) {
56        dinoTL[j] = dinoHitL[j];
57        dinoTR[j] = dinoHitR[j];
58      } else {
59        dinoTL[j] = dinoTL1[j];
60        dinoTR[j] = dinoTR1[j];
61      }
62      dinoBL[j] = dinoBL1[j];
63      dinoBR[j] = dinoBR1[j];
64    }
65    dino_sprite_change = true;
66  }
67
68  void DinoController::SetWalk2() {
69    int j;
70    for (j = 0; j < 8; j++) {
71      if (dino_is_hit) {
72        dinoTL[j] = dinoHitL[j];
73        dinoTR[j] = dinoHitR[j];
74      } else {
75        dinoTL[j] = dinoTL1[j];
76        dinoTR[j] = dinoTR1[j];
77      }
78      dinoBL[j] = dinoBL2[j];
79      dinoBR[j] = dinoBR2[j];
80    }
81    dino_sprite_change = true;
82  }
83
84  void DinoController::SetJumpCrouch() {
85    if (dino_is_hit) {
86      return;
87    }
88    int j;
89    for (j = 0; j < 8; j++) {
90      dinoTL[j] = JumpCrouch1[j];
91      dinoTR[j] = JumpCrouch2[j];
92      dinoBL[j] = JumpCrouch3[j];
93    }
94    dino_sprite_change = true;
95  }
```

```cpp
 97 void DinoController::Jump() {
 98   if (dino_is_hit) {
 99     return;
100   }
101   SetJumpCrouch();
102   dino_state = DINO_STATE_JUMP;
103   dino_state_duration = 1.25;
104 }
105
106 void DinoController::Crouch() {
107   if (dino_is_hit) {
108     return;
109   }
110   SetJumpCrouch();
111   dino_state = DINO_STATE_CROUCH;
112   dino_state_duration = 1.25;
113 }
114
115 void DinoController::NextState() {
116   switch (dino_state) {
117     case DINO_STATE_WALK1_TRANSITION:
118       SetWalk1();
119       dino_state = DINO_STATE_WALK1;
120       dino_state_duration = 0.250;
121       break;
122     case DINO_STATE_WALK1:
123       dino_state = DINO_STATE_WALK2_TRANSITION;
124       dino_state_duration = 0;
125       break;
126     case DINO_STATE_WALK2_TRANSITION:
127       SetWalk2();
128       dino_state = DINO_STATE_WALK2;
129       dino_state_duration = 0.250;
130       break;
131     case DINO_STATE_WALK2:
132       dino_state = DINO_STATE_WALK1_TRANSITION;
133       dino_state_duration = 0;
134       break;
135     case DINO_STATE_CROUCH:
136     case DINO_STATE_JUMP:
137       dino_state = DINO_STATE_WALK1_TRANSITION;
138       dino_state_duration = 0;
139       break;
140
141   }
142
143   dino_state_time = 0;
144 }
```

WeedController.h

```
1  #ifndef WeedController_h
2  #define WeedController_h
3
4  #define WEED_STATE_WALK1_TRANSITION 1
5  #define WEED_STATE_WALK1 2
6
7  #define WEED_STATE_WALK2_TRANSITION 3
8  #define WEED_STATE_WALK2 4
9
10 #define WEED_STATE_WALK3_TRANSITION 5
11 #define WEED_STATE_WALK3 6
12
13 #define WEED_STATE_WALK4_TRANSITION 7
14 #define WEED_STATE_WALK4 8
15
16 #include "Arduino.h"
17 #include "LiquidCrystal_I2C.h"
18
19 class WeedController {
20   private:
21     static byte frame1[8] = {
22       0b00000,
23       0b00000,
24       0b00110,
25       0b01101,
26       0b10111,
27       0b11011,
28       0b01110,
29       0b01100
30     };
31
32     static byte frame2[8] = {
33       0b00000,
34       0b00000,
35       0b00000,
36       0b00000,
37       0b10110,
38       0b11101,
39       0b11011,
40       0b00110
41     };
42
43     static byte frame3[8] = {
44       0b00000,
45       0b00000,
46       0b00000,
47       0b01010,
48       0b11100,
49       0b10111,
50       0b01111,
51       0b00110
52     };
53
54     static byte frame4[8] = {
55       0b00000,
56       0b00000,
57       0b11100,
58       0b10110,
59       0b01111,
60       0b11011,
61       0b01110,
62       0b00000
63     };
64
65
66     int weed_state;
67     float weed_state_time;
68     float weed_state_duration;
69
70     void NextState();
71     void SetFrame(int num);
72
73   public:
74
75     byte frame[8];
76     bool weed_sprite_change;
77
78
79     void Init();
80     void Update(float s);
81
82 };
83
84 #endif
```

WeedController.cpp

```cpp
1  #include "Arduino.h"
2  #include "WeedController.h"
3
4  // needs to be define here with the actual value in the h file
5  static byte WeedController::frame1[8];
6  static byte WeedController::frame2[8];
7  static byte WeedController::frame3[8];
8  static byte WeedController::frame4[8];
9
10 void WeedController::Init() {
11   weed_state = WEED_STATE_WALK1_TRANSITION;
12   weed_state_time = 0;
13   weed_state_duration = 0;
14   SetFrame(0);
15 }
16
17 void WeedController::Update(float s) {
18   weed_state_time += s;
19   if (weed_state_time >= weed_state_duration) {
20     NextState();
21   }
22 }
23
24 void WeedController::SetFrame(int num) {
25   int j;
26   switch (num) {
27     case 0:
28       for (j = 0; j < 8; j++) {
29         frame[j] = frame1[j];
30       }
31       break;
32     case 1:
33       for (j = 0; j < 8; j++) {
34         frame[j] = frame2[j];
35       }
36       break;
37     case 2:
38       for (j = 0; j < 8; j++) {
39         frame[j] = frame3[j];
40       }
41       break;
42     case 3:
43       for (j = 0; j < 8; j++) {
44         frame[j] = frame4[j];
45       }
46       break;
47   }
48   weed_sprite_change = true;
49 }
```

```
51 void WeedController::NextState() {
52   switch (weed_state) {
53     case WEED_STATE_WALK1_TRANSITION:
54       SetFrame(0);
55       weed_state = WEED_STATE_WALK1;
56       weed_state_duration = 0.250;
57       break;
58     case WEED_STATE_WALK1:
59       weed_state = WEED_STATE_WALK2_TRANSITION;
60       weed_state_duration = 0;
61       break;
62     case WEED_STATE_WALK2_TRANSITION:
63       SetFrame(1);
64       weed_state = WEED_STATE_WALK2;
65       weed_state_duration = 0.250;
66       break;
67     case WEED_STATE_WALK2:
68       weed_state = WEED_STATE_WALK3_TRANSITION;
69       weed_state_duration = 0;
70       break;
71     case WEED_STATE_WALK3_TRANSITION:
72       SetFrame(2);
73       weed_state = WEED_STATE_WALK3;
74       weed_state_duration = 0.250;
75       break;
76     case WEED_STATE_WALK3:
77       weed_state = WEED_STATE_WALK4_TRANSITION;
78       weed_state_duration = 0;
79       break;
80     case WEED_STATE_WALK4_TRANSITION:
81       SetFrame(3);
82       weed_state = WEED_STATE_WALK4;
83       weed_state_duration = 0.250;
84       break;
85     case WEED_STATE_WALK4:
86       weed_state = WEED_STATE_WALK1_TRANSITION;
87       weed_state_duration = 0;
88       break;
89   }
90
91   weed_state_time = 0;
92 }
```

Bill of Materials

1602 LCD 16x2 Character Display w/ I2C Controller - $2.00 – 1 Req.

https://www.aliexpress.com/item/1PCS-LCD-module-Blue-screen-IIC-I2C-1602-for-arduino-1602-LCD-UNO-r3-mega2560/32763867041.html
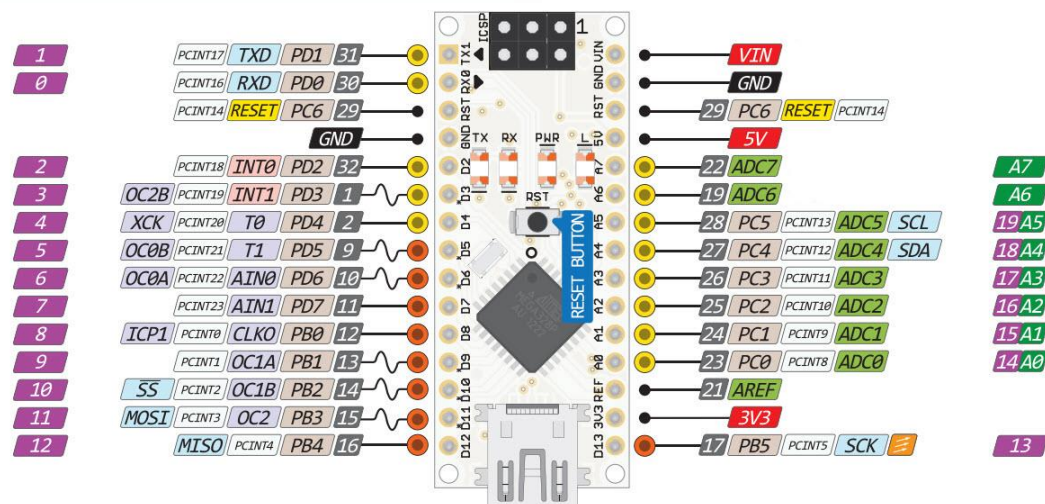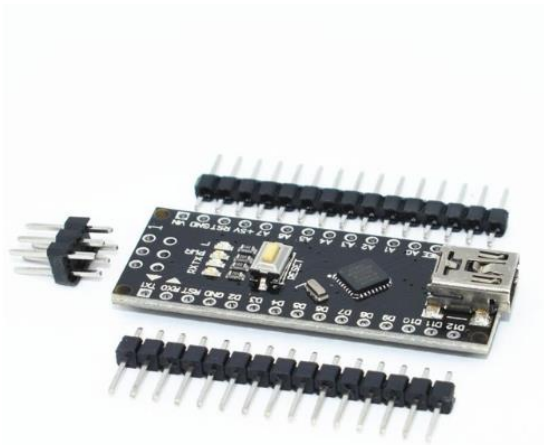

If you hook up this display and all you get are white blocks on the top row, you have likely configured the wrong address in your code.

See RetroLCD.com for a helpful sketch that will tell you what address your display is listening on.

Arduino Nano Clone - $1.90 – 1 Req.

https://www.aliexpress.com/item/Freeshipping-Nano-3-0-controller-compatible-for-arduino-nano-CH340-USB-driver-NO-CABLE/32341832857.html

Note: This board is recognized as an Ardunio Duemilanove or Diecimila, ATmega 328P. If your IDE has trouble uploading, it may be because you've selected the wrong board variant. The Duemilanove is the version of the Arduino before the UNO. To test the board, simply plug it into your USB port and view the serial monitor. It will spit out all the ASCII character codes.

74HX4051N DIP-16 Multiplexer - $2.04 / 10, $0.204 each – 1 Req.

https://www.aliexpress.com/item/10pcs-free-shipping-74HC4051N-74HC4051-SN74HC4051N-DIP-16-Multiplexer-Switch-ICs-8-CHANNEL-ANALOG-MUX-DEMUX/32416713940.html

Ceramic Capacitor - $1.40 / 300, $0.0047 each – 1 Req.

https://www.aliexpress.com/item/Ceramic-capacitor-2PF-0-1UF-30-valuesX10pcs-300pcs-Electronic-Components-Package-ceramic-capacitor-Assorted-Kit-Free/32305092269.html

The rating really doesn't matter. In fact, this part is probably optional.

Resistor - $2.48 / 600, $0.0041 each – 1 Req.

https://www.aliexpress.com/item/Free-Shipping-600-Pcs-1-4W-1-20-Kinds-Each-Value-Metal-Film-Resistor-Assortment-Kit/32323198194.html

This resistor is used to pull down the buttons when they're not pressed so that you don't get invalid button pushes. This is also handled in code to ensure when a button is let go, the Multiplexer doesn't think it is still pressed. 220ohm is what I use but there is no strict requirement.

Tactile Push Button Switch 12x12x4.3mm – $1.79 / 50, $0.0358 each – 4 Req.
https://www.aliexpress.com/item/R242-03-12-12-4-3MM-touch-switch-micro-switch-vertical-feet-4/32691509241.html

The PCB supports up to 8 buttons. 4 are required for Dinosaur.

Pin Header Connector Male 2.54mm Pitch Single Row 40 Pin - $1.85 / 30 x 40, $0.0015 each – 16 Req.

https://www.aliexpress.com/item/McIgIcM-60PCS-1-x-40-Pin-2-54mm-Spacing-Single-Row-Breakable-Male-Pin-Header-Connector/32809323787.html
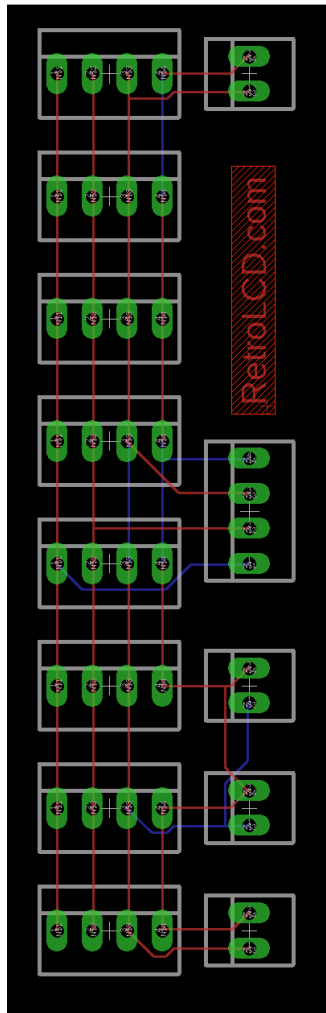
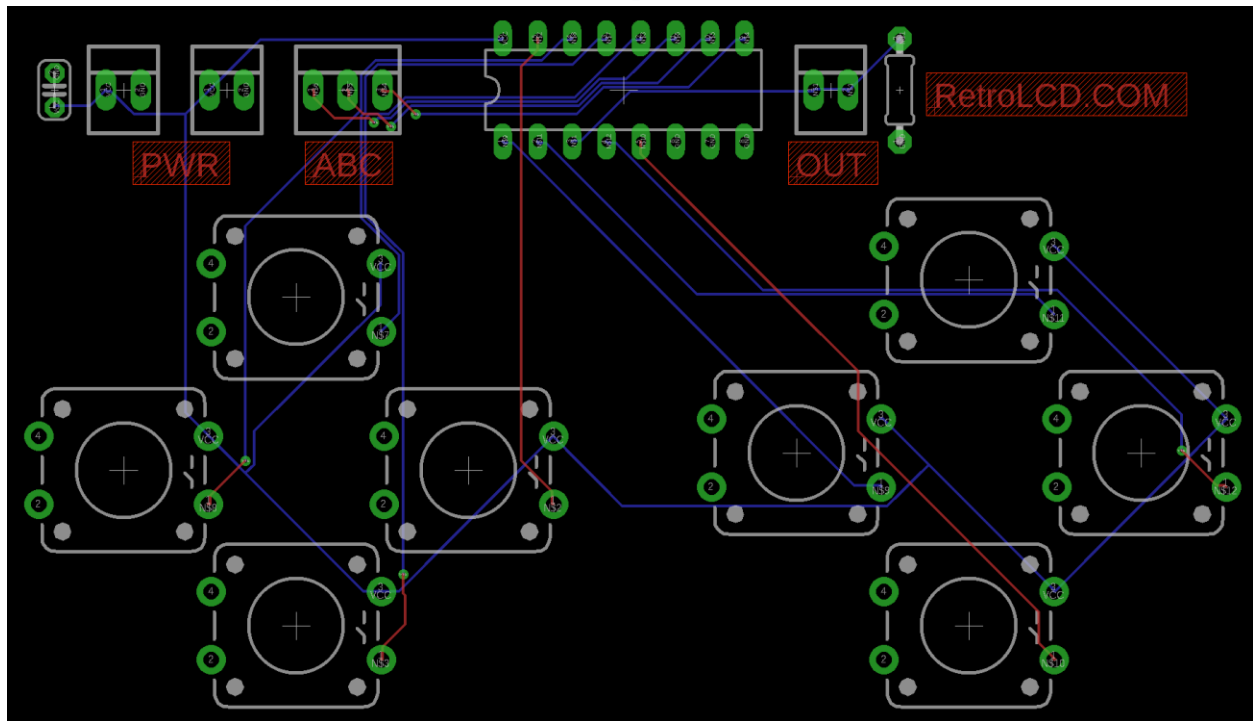Female – Female Jumper Wire 20cm 2.54mm 1p-1p – $0.78 / 40, $0.0195 each – 14 Req

https://www.aliexpress.com/item/Free-Shipping-80pcs-dupont-cable-jumper-wire-dupont-line-female-to-female-dupont-line-20cm-1P/1728848121.html

I2C / Power Hub - $2.00 / 10 - $0.20 each – 1 Req.

RetroLCD.com

Controller - $2.00 / 10 - $0.20 each – 1 Req.

Jumbo Wood Craft Sticks - $5.30 / 200 – $0.0265 each – 5 Req.
https://www.michaels.com/creatology-jumbo-wood-craft-sticks/10334892.html

RetroLCD.com

Summary

Part	Price	Units	Per Unit	Required	Total
1602 LCD 16x2 Character Display w/ I2C Controller	$2.00	1	$2.0000	1	$2.0000
Arduino Nano Clone	$1.90	1	$1.9000	1	$1.9000
74HX4051N DIP-16 Multiplexer	$2.04	10	$0.2040	1	$0.2040
Ceramic Capacitor	$1.40	300	$0.0047	1	$0.0047
Resistor	$2.48	600	$0.0041	1	$0.0041
Tactile Push Button Switch 12x12x4.3mm	$1.79	50	$0.0358	4	$0.1432
Pin Header Connector Male 2.54mm Pitch Single Row 40 Pin	$1.85	1200	$0.0015	16	$0.0240
Female – Female Jumper Wire 20cm 2.54mm 1p-1p	$0.78	40	$0.0195	14	$0.2730
Jumbo Wood Craft Sticks	$5.30	200	$0.0265	5	$0.1325
I2C / Power Hub	$2.00	10	$0.2000	1	$0.2000
Controller	$2.00	10	$0.2000	1	$0.2000
Total	$23.54				$5.0855

* Prices are accurate at a point in time and are subject to change – every effort is made to choose generic parts that have little risk of going out of production