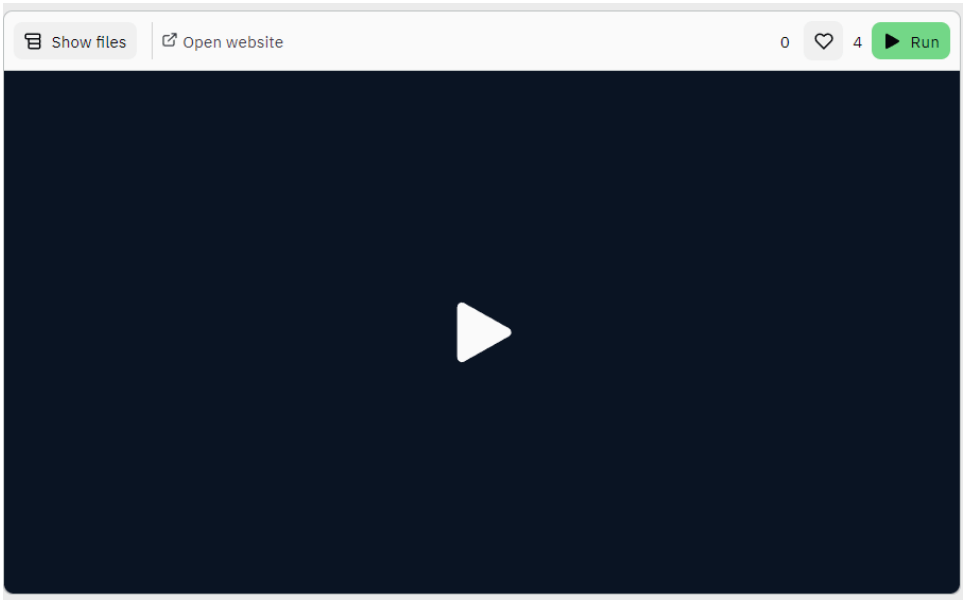
DISCLAIMER: This guide was created for the Build a Game workshop with Girls Who Code at UNC Charlotte, developed by Ashley Harmon. The steps in this guide are following the first section of the Easy JavaScript Development with Kaboom.js course by Ania Kubów, with updates to accommodate the changes with Kaboom and Replit since that video was made. The steps were typed up for easy following along after our live workshop together, and some extra tips and resources have been added in this guide for those wishing to learn more! Check out Ania Kubów for more awesome tutorials.



Introduction

In this workshop, we will be using Replit (a browser-based IDE) and Kaboom.js (a JavaScript library). If you haven't heard of an IDE before, you can read more here and explore some other cool browser-based IDEs. Kaboom.js will allow us to create games with a lot less work. You can refer to its documentation here if you get stuck or want to try more things.

If you would like to follow along with the workshop recording, stay tuned for that link! You can also test out the game or see the game code here. When you see the preview window, you can click the **Run** button (top right corner of the preview section) to play the game or the **Show files** button (top left corner of the preview section) to see the code. All of the code developed in this workshop is in the **code/main.js** file.



Basic Coding Concepts

If you are new to using coding languages in general, there are some concepts explained here. We won't go over these today, but if you are interested in understanding the code we use a bit more, it's always great to spend some time looking into the fundamentals. Kaboom's documentation will have more information on each of the methods and functions used today.

Getting started in Replit

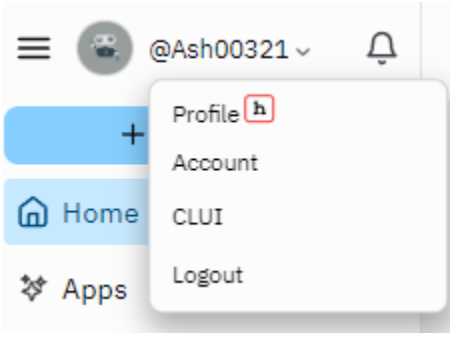
(Optional) Creating an account

1) Go to https://replit.com/
2) Use the **Sign Up** button to create an account
   If you have a GitHub account, you can use the **Continue with GitHub** option.

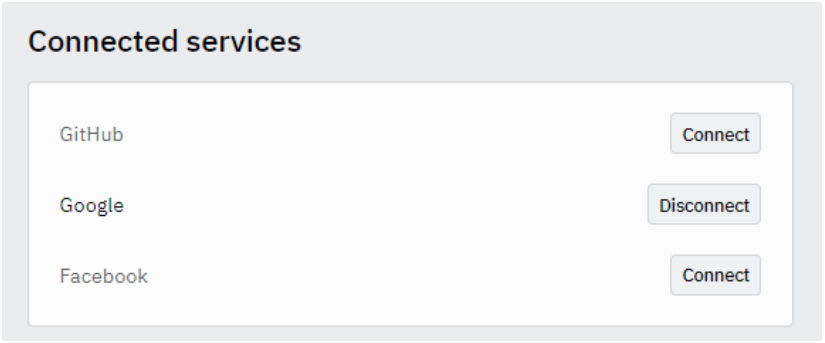
(Optional) Connecting to GitHub

If you already have an account and would like to connect it to GitHub, you can use the steps below. Connecting it will allow you to keep a copy of your project as a repo in GitHub that you can keep updated with all your changes. This is recommended if you would like to use other IDEs in the future, such as VS Code, and it is never a bad idea to have all your projects stored on GitHub.

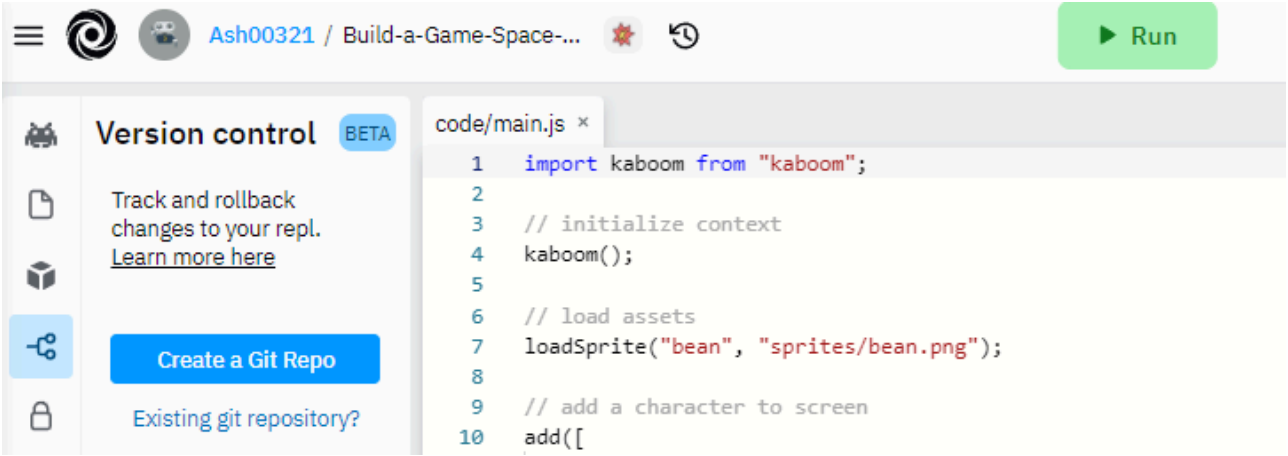
1) Make sure you are signed in on https://replit.com/
2) Click on the **down arrow** next to your username in the top left corner and select **Account**



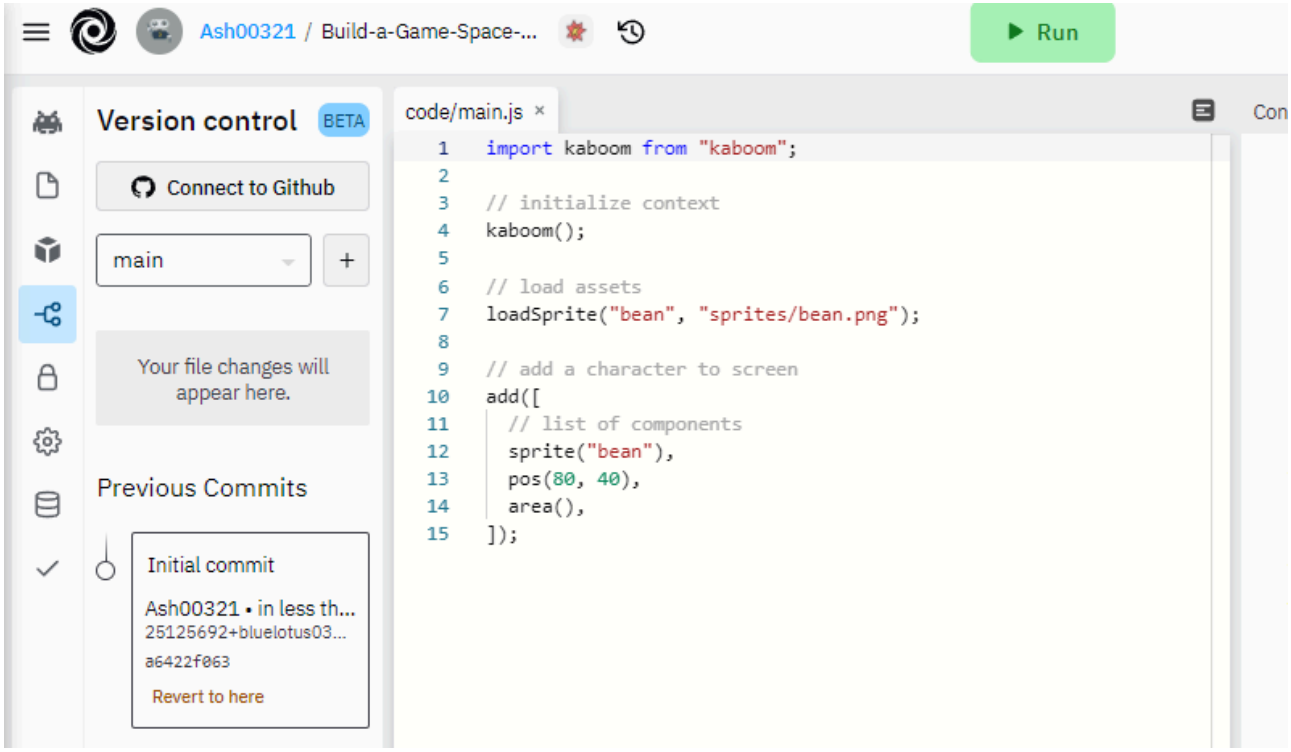
3) Scroll down to **Connected Services** and click the **Connect** button next to GitHub



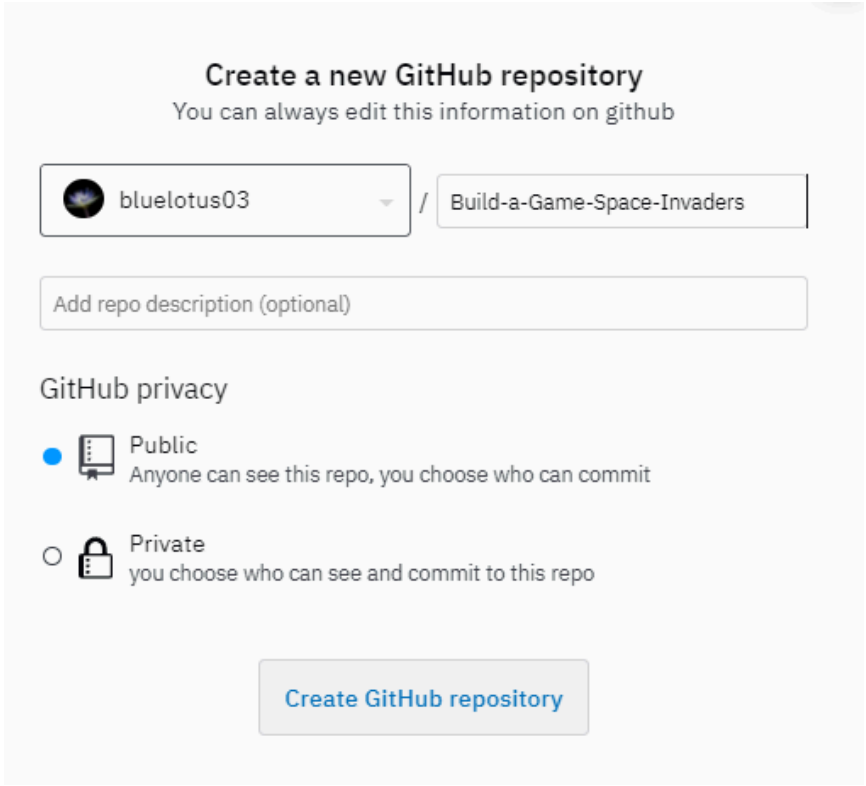
4) Now, once you have a repl (project) created, you can select the **Version control** tab on the left side panel and click the **Create a Git Repo** button



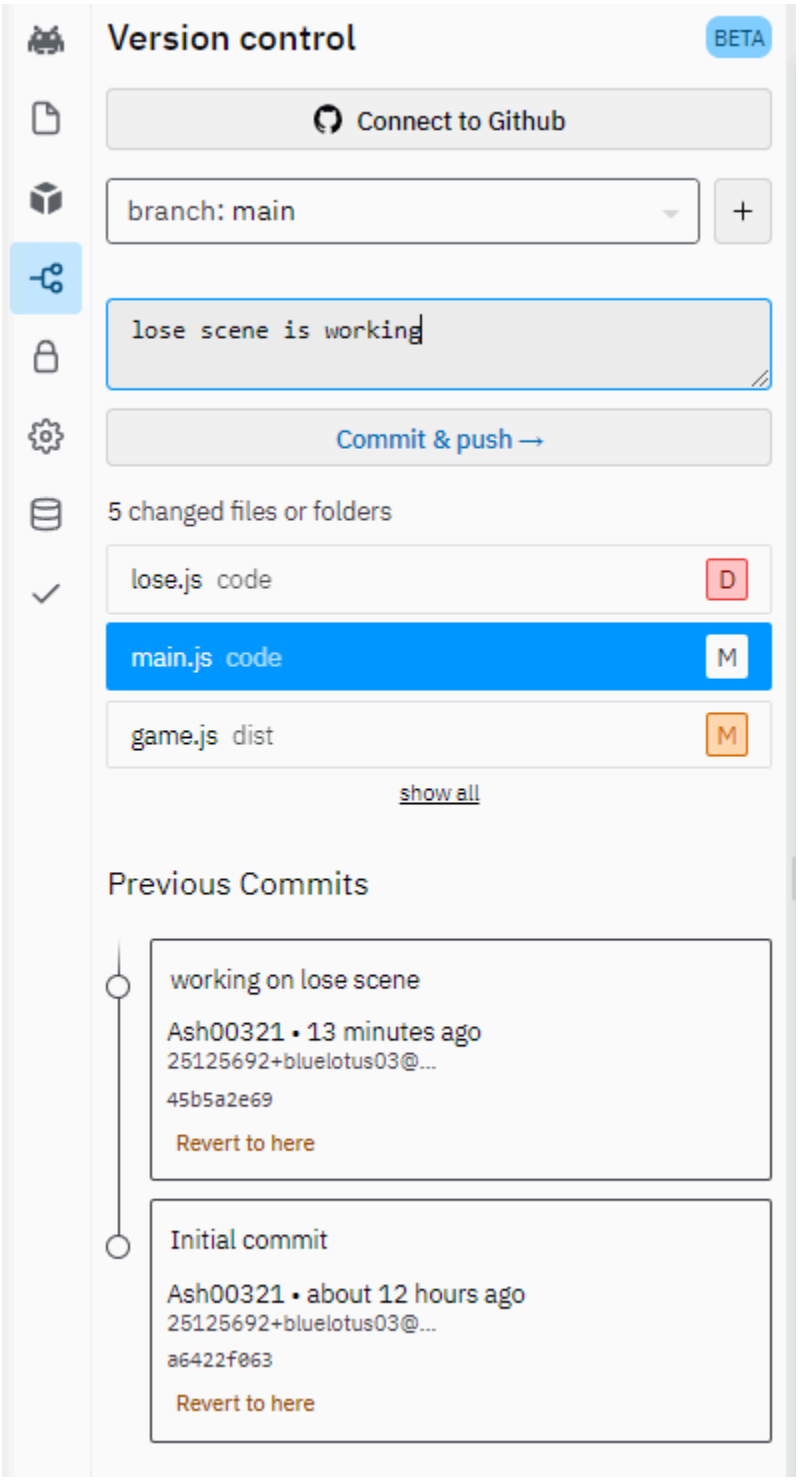
5) Click the **Connect to Github** button



6) You can check to see that it has your Github username and the name of the repo. You can add a description for the repo as well if you'd like. Choose if you want it to be private or public, and click the **Create GitHub repository** button.

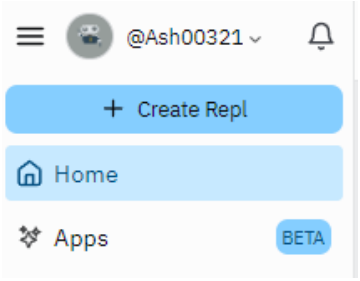


7) Whenever you want to save your changes to GitHub, you can use the **Version control** tab on the left side panel, type in a commit message (noting what you changed), and click the **Commit & push** → button

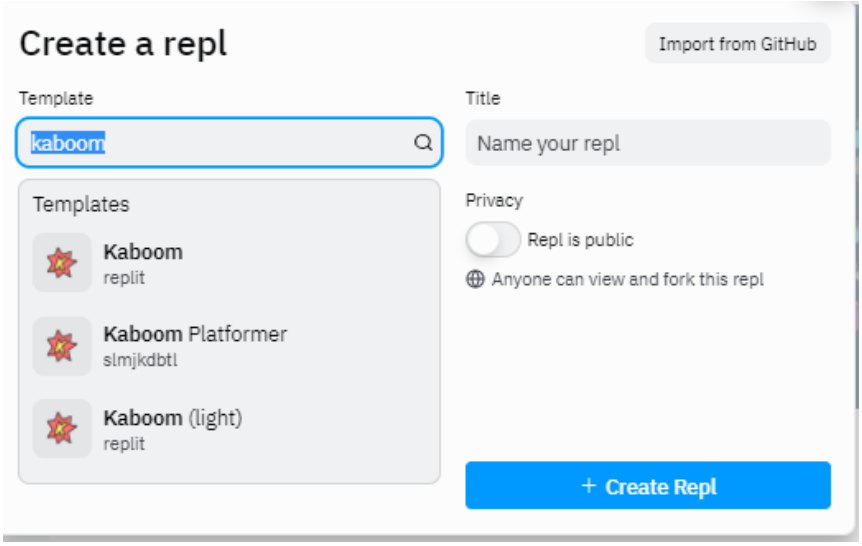


Starting a Kaboom.js Project

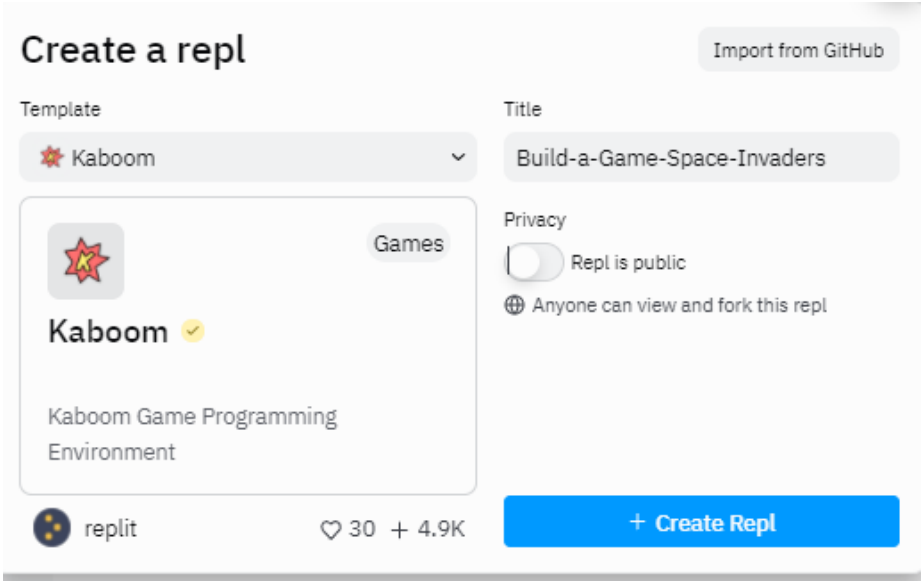
1) Start a new project by clicking the **+ Create Repl** button in the top left corner of your homepage



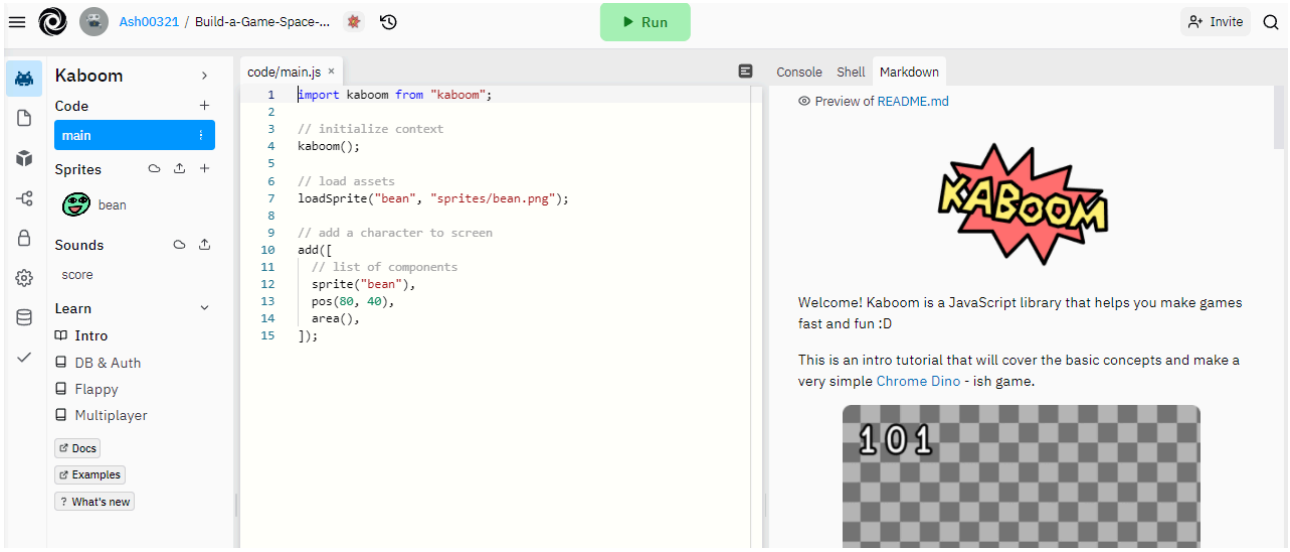
2) Search for **kaboom** and select **Kaboom replit** (first option)



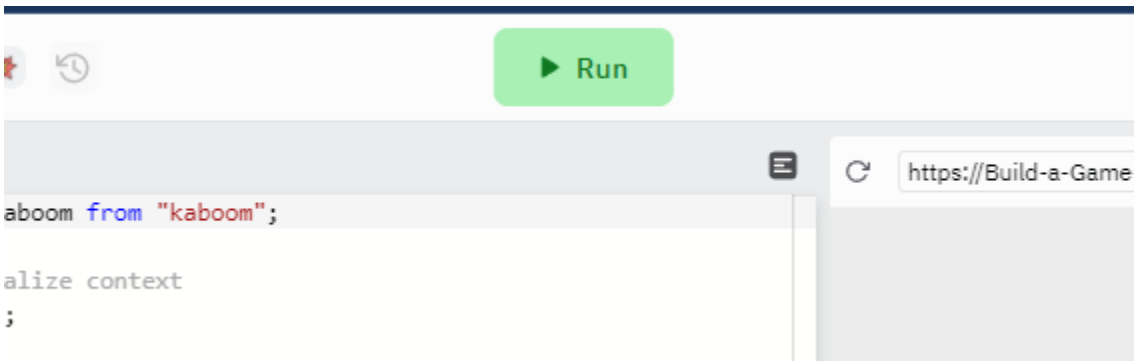
3) Give your repl a **title**, you can leave it as **public**, and click the **+ Create Repl** button



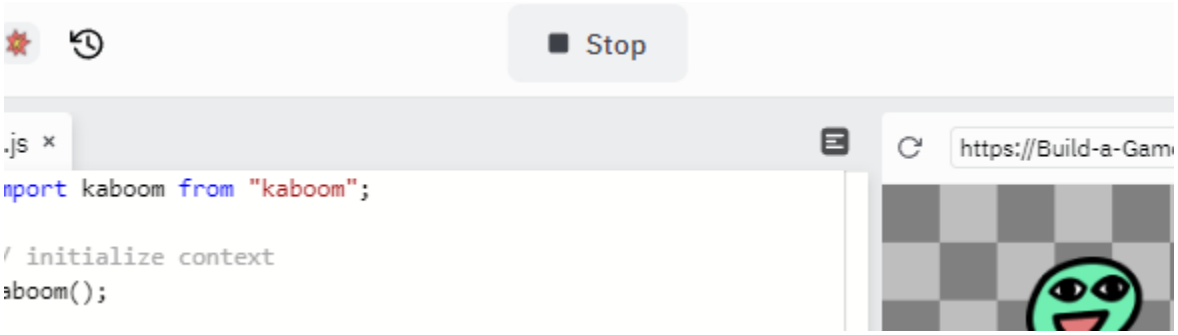
4) You should see the following setup (or similar - depending on when you are doing this workshop)



5) The template for this kaboom repl has some things already loaded for us, and if we click the **Run** button, we can see the bean sprite (image) on a game board. This Run button is how we will run our code and check to see how the changes we have made in the code affect what appears on screen.

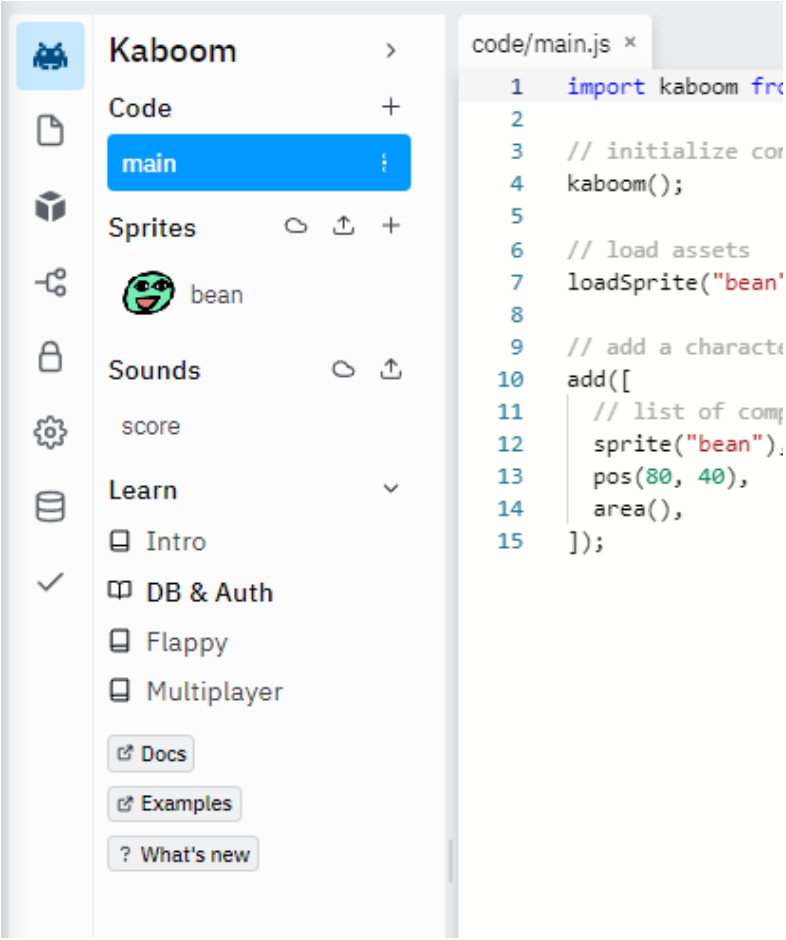


6) You can click the **Stop** button to stop running the code, which we will go ahead and do for now. However, you can leave it running and see changes reflected just by using **CTRL + S** to save the file in which you have made changes to the code.



NOTE: The video this workshop is based on uses Replit's version 0.5.0 of Kaboom.js, but that is no longer available or shown as an option to change on Replit. We will use Replit's version 2000.1.1, which is the default for the kaboom repl we chose earlier. However, if you are completing this workshop outside of the live event, you may be using a different version if Replit has updated things again. If something doesn't work as it should somewhere down the line (and it is not an error in your code), it is likely due to using a changed version.You can look at the Replit documentation and the Kaboom documentation to see what has changed since this workshop and adjust accordingly. Here is a link to Replit's GitHub repo for kaboom as well. Inside the Replit project, you will also see a **? What's new** button under the **Kaboom** tab.

Here is a snapshot of the Kaboom.js documentation at the time this workshop took place. If any functions have been removed since then, you can view them on here to see what they did and get a better idea of what to search for in hopes of finding the function that has replaced it.



Sprites

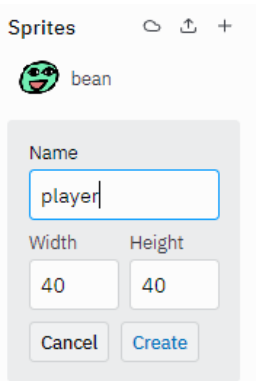
Creating Sprites

A sprite is essentially a 2-dimensional image or animation used in a game. This could be a character, an object, etc., moving independently from the background. Replit lets us create sprites right here in the IDE! The Kaboom.js repl has given us a default sprite, which you are free to use if you'd like, but we will go ahead and make our own sprite now.

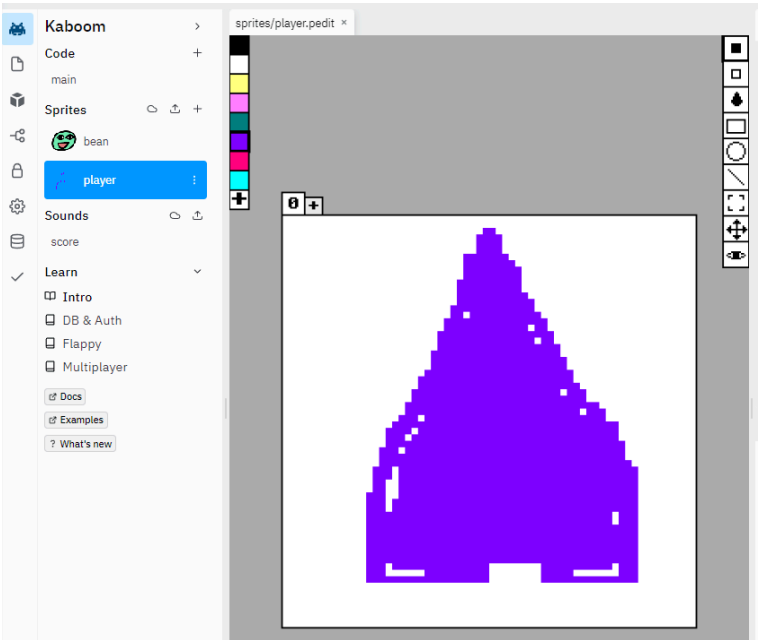
1) Click the **+** button next to **Sprites**



2) Give the sprite a **name** (we'll start with "player"), a **width** and **height** (in this case 40x40), and click **Create**



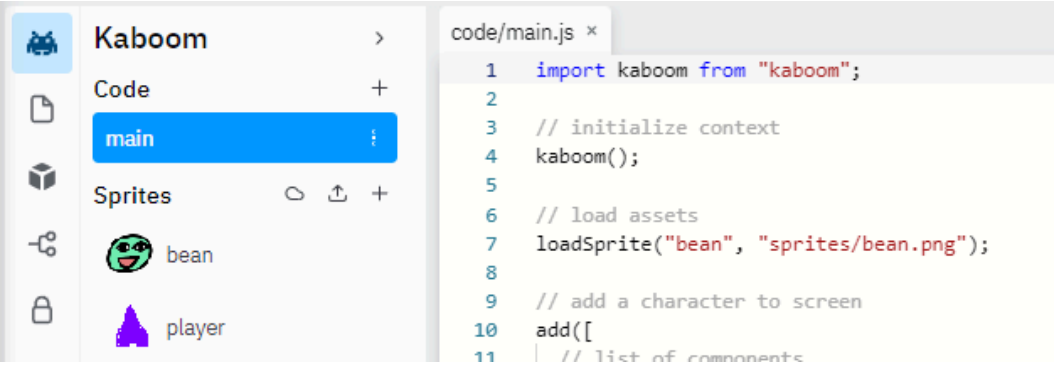
3) You should see a blank canvas, color option, and some tools on the right hand side. You can hover over a tool to see the name of it and get an idea of what it does. By default, the pen tool should be selected (black rectangle), and you can go ahead and pick the color on the left hand side and start drawing! Since we are making our own version of Space Invaders, I will have the player be a little plane. (no - I'm not an artist haha)



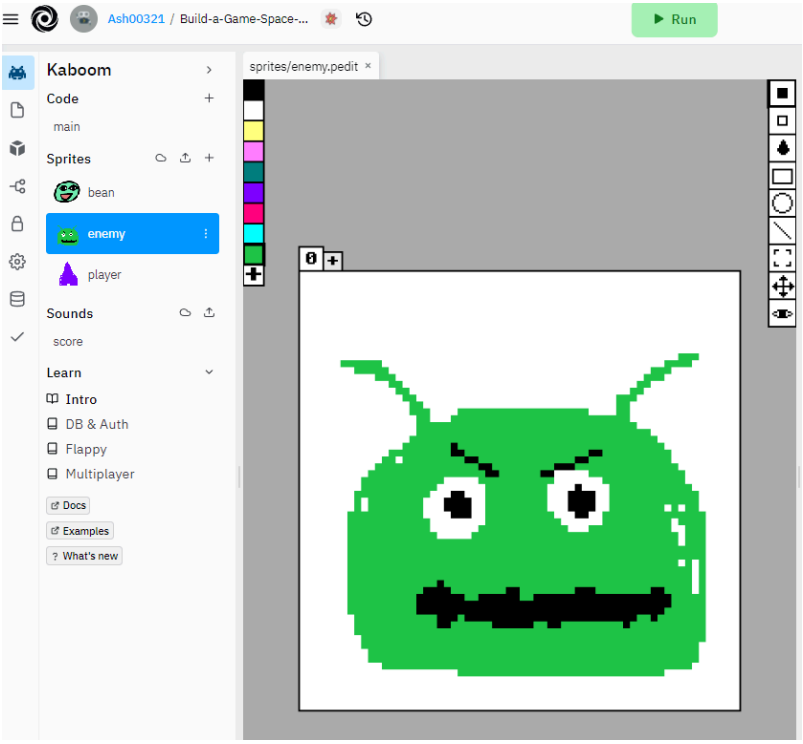
4) Click the **save** button (floppy disk icon) in the bottom right panel



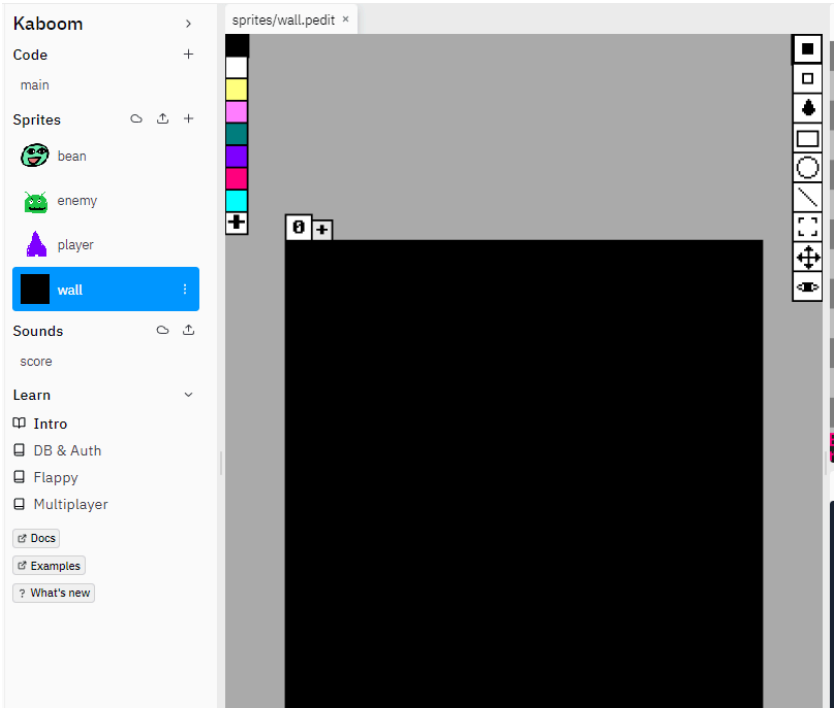
5) You can make sure it was saved properly by clicking on **main**. You should be able to see your new sprite under the **Sprites** section



6) Now we can repeat these steps to create our enemy sprite. In this case, I have made the enemy a little alien creature for our Space Invaders game. If you'd like your enemies to be a bit smaller than the player, you can leave the default width and height (32x32).



7) We will repeat these steps one more time to create our wall for the game borders, in this case, a solid black block. We have set the width and height to be 30x22.



Loading Sprites into the Game

In scenes/main.js, we can load our sprites easily by hovering over the sprite in the left side panel, clicking the **3 dots**, and selecting **Insert load code**. This will insert it wherever your cursor is currently in the main.js file, so make sure you are on the line you want it to load in. You can remove line 7, as we will not use the bean.

1) Start by putting your cursor on line 7, hovering over the **player** sprite, clicking the **3 dots**, and select **Insert load code**

```
6   // load assets
7   loadPedit("player", "sprites/player.pedit");
```

2) Change “bean” to “player” on line 12

```
9   // add a character to screen
10  add([
11      // list of components
12      sprite("player"),
13      pos(80, 40),
14      area(),
15  ]);
```

3) Click **Run** and see your player sprite on the board now!

About the code:
This add() function is currently using the sprite() function to render our player sprite, the pos() function to put the player sprite in a specific position on the screen with x and y coordinates (80 pixels from the left and 40 pixels from the top), and the area() function to generate a collider area that allows us to use the collision detection later if our player collides with an enemy.

If you'd like to scale your sprite up or down, you can also add the scale() function and enter the amount you want to scale it. For example, to shrink our sprite to half its original size, you could use scale(0.5).

4) Delete the add() function on line 10 for now, we will add our player back after the level creation

5) We will repeat step 1 to load the enemy and wall sprites as well
Adding them with their positions will come later

```
6   // load assets
7   loadPedit("player", "sprites/player.pedit");
8   loadPedit("enemy", "sprites/enemy.pedit");
9   loadPedit("wall", "sprites/wall.pedit");
10
```

Add a Level, Player, and More

You'll notice that we currently just have a checkerboard looking default background. We want to change this to have some walls to keep gameplay in a certain area, and eventually a solid black background color to simulate being in outer space.

1) Use the **addLevel() function** to specify what should be on the screen (the map) and the configuration settings of items on the map (such as width and height).

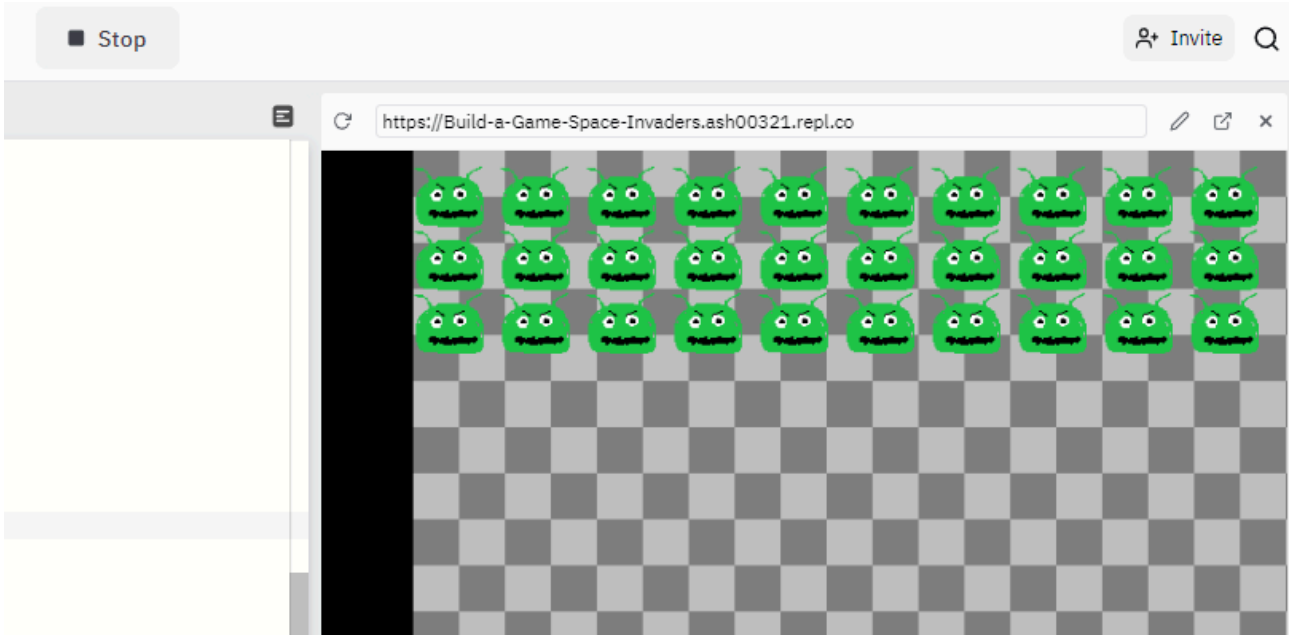
About the code:
This addLevel() function is currently using the first argument (in the square brackets [ ]) to specify the layout of the map, and it has the second argument (in curly braces { }) specifying information about the items on the map.

In the square brackets, we can use symbols to notate specific objects (like an enemy or background). Then in the curly braces, we can state what those symbols represent and some information about their configuration, such as width and height.

Here we are using the ! for the left wall, the & for the right wall, the ^ for the enemies, and x for the player. We have also scaled down the enemies just slightly to keep them from taking up a full square. This gives some better spacing between them.

```
31  //add a level
32  addLevel([
33    '!! ^^^^^^^^^^^^^^^^^^^     &&',
34    '!!                         &&',
35    '!! ^^^^^^^^^^^^^^^^^^^     &&',
36    '!!                         &&',
37    '!! ^^^^^^^^^^^^^^^^^^^     &&',
38    '!!                         &&',
39    '!!                         &&',
40    '!!                         &&',
41    '!!                         &&',
42    '!!                         &&',
43    '!!                         &&',
44    '!!                         &&',
45    '!!                         &&',
46    '!!                         &&',
47    '!!                         &&',
48    '!!                         &&',
49    '!!                         &&',
50    '!!                         &&',
51  ], {
52    width: 30,
53    height: 22,
54    '^': () => [sprite('enemy'), 'enemy', scale(0.5), area
      ()],
55    '!': () => [sprite('wall'), 'left-wall',  area()],
56    '&': () => [sprite('wall'), 'right-wall',  area()],
57  })
58
```

2) **Save** your file, and you should see the following changes

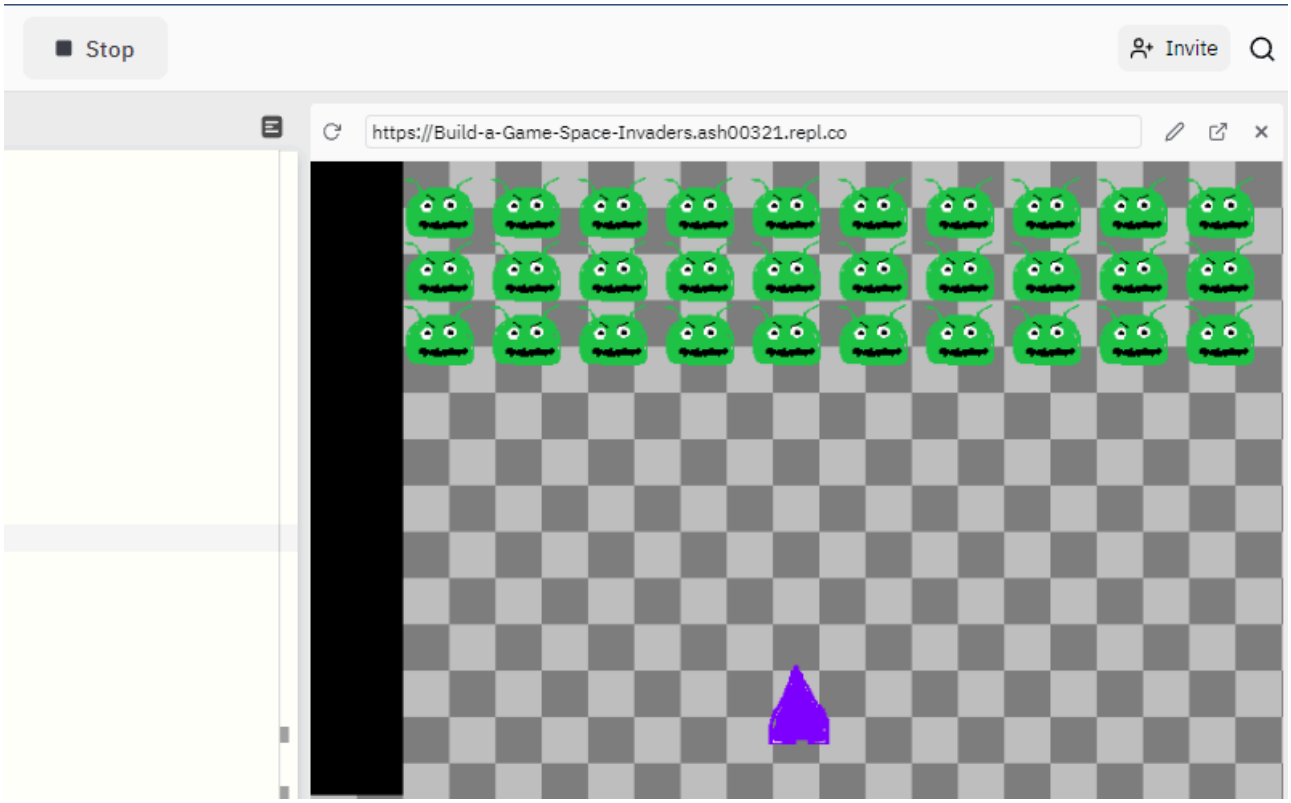


Add the Player

1) Use the **add() function** again, but set it equal to the constant **player**

```
61
62  // add the player
63  const player = add([
64    sprite('player'),
65    scale(0.9),
66    pos(width() / 2, height() / 1.2),
67    origin('center'),
68    area()
69  ])
70
```

2) Save your file, and you should see the following changes



Add Keystroke Events

We can specify what we want to happen when the player presses certain keys on their keyboard using the keyDown() function, which may have changed in recent versions. In our case, we want the player to be able to move left and right, as well as shoot.

1) Set a **constant to save the speed** with which we want the player to be able to move

```
51
52    // set move speed
53    const MOVE_SPEED = 200
54
```

2) Use the **keyDown() function** to specify the player should move left when pressing the left arrow key

```
55    // add keystroke events
56    keyDown('left', () => {
57      player.move(-MOVE_SPEED, 0)
58    })
```

3) Use the **keyDown() functio**n to specify the player should move right when pressing the right arrow key

```
60    keyDown('right', () => {
61      player.move(MOVE_SPEED, 0)
62    })
```

4) **Save** and test out your left and right arrow keys!

Add Enemy Motion

We want our enemies to move left and right, so we will use the onUpdate() function to specify how we want them to do this.

1) **Set a constant** to save the speed with which we want the enemies to move
Add a current speed for the enemy, as well as the speed to move down when colliding with a wall. We can add this to our //set move speeds section

```
// set move speeds
const MOVE_SPEED = 200
const ENEMY_SPEED = 125
let CURRENT_SPEED = ENEMY_SPEED
```

2) Use the **onUpdate() function** to specify how the enemy should move with this speed

```
86    // add enemy motion
87    onUpdate('enemy', (e) => {
88      e.move(CURRENT_SPEED, 0)
89    })
90
```

3) Use the **collides() function** to specify what happens when the enemies hit our right wall

```
const LEVEL_DOWN = 100

// enemy collides with right wall
onCollide('enemy', 'right-wall', () => {
    CURRENT_SPEED = -ENEMY_SPEED
    every('enemy', (e) => {
      e.move(0, LEVEL_DOWN)
    })
})
```

4) Use the **collides() function** to specify what happens when the enemies hit our left wall

```
// enemy collides with left wall
onCollide('enemy', 'left-wall', () => {
    CURRENT_SPEED = ENEMY_SPEED
    every('enemy', (e) => {
      e.move(0, LEVEL_DOWN)
    })
})
```

5) Test it out! Click **Run**, and you should see your enemies moving left and right within the walls, and moving down when hitting a wall

Add a Score

1) Use the **layer() function** to separate the objects on the board from the score we will add
We will add this above the addLevel() function

```
layer(['obj', 'ui'], 'obj')

addLevel([
  '! ^ ^ ^ ^ ^ ^ ^ ^ ^ ^     &',
```

2) Create a **constant to hold the score**, and use the **add() function** to add it to the game
We can add this above the const player

```
// add the score
const score = add([
  text('0'),
  pos(5,5),
  layer('ui'),
  {
    value: 0,
  }
])
```

3) We can add a bit more to our side walls by adding one more left and right wall on our map This will allow for more space on the left side for the score, though you can always scale the score down if you'd like

```
addLevel([
  '!! ^ ^ ^ ^ ^ ^ ^ ^ ^ ^     &&',
  '!!                         &&',
  '!! ^ ^ ^ ^ ^ ^ ^ ^ ^ ^     &&',
  '!!                         &&',
  '!! ^ ^ ^ ^ ^ ^ ^ ^ ^ ^     &&',
  '!!                         &&',
  '!!                         &&',
  '!!                         &&',
  '!!                         &&',
  '!!                         &&',
  '!!                         &&',
  '!!                         &&',
  '!!                         &&',
  '!!                         &&',
  '!!                         &&',
  '!!                         &&',
], {
  width: 30,
  height: 22,
  '^': () => [sprite('enemy'), 'enemy', scale(0.9), area()],
  '!': () => [sprite('wall'), 'left-wall',  area()],
  '&': () => [sprite('wall'), 'right-wall',  area()],
})
```

4) Try it out! Click the **Run** button to see your initial score of 0 on the top left of the screen

Losing Conditions

Create a Lose Scene

We want to tell the game what to do if the player loses the game.

1) Use the **scene() function** to specify what should show on our lose screen
   We can add this right below the layer() function, and in this case, we will print the words "Game Over"

```
13  scene('lose', () => {
14    add([
15      text('Game Over'),
16      origin('center'),
17      pos(width() / 2, height() / 2)
18    ])
19  })
20
```

Create the Lose Scenario

If the enemies move down far enough to collide with the player, the game should end in a loss. We want to also specify that the game is over if the enemies reach the bottom of the game board, rather than allowing them to continue moving around the player down the entire screen

1) Use the **onCollide()** function to specify that the player loses the game if they encounter an enemy directly

```
103  // player collides with enemy
104  player.onCollide('enemy', () => {
105    go('lose')
106  })
```

2) Use the **onUpdate()** function to specify what happens when the enemies reach the bottom of the game board

```
134  // enemies reach bottom of game board
135  onUpdate('enemy', (e) => {
136    if (e.pos.y >= height() - 70) {
137      go('lose')
138    }
139  })
140
```

Shooting

Add Shooting Capabilities

We want our player to be able to shoot at the enemies using the spacebar.

1) Create a **function to spawn the bullets**

```
// add bullets
function spawnBullet(p) {
  add([
    rect(3, 5),
    pos(p),
    origin('center'),
    color(255, 8, 246),
    area(),
    'bullet'
  ])
}
```

2) Use the **keyDown() function** to specify the spacebar as the “shoot” button
We can add this right below our other keyDown events

```
92
93    keyDown('space', () => {
94      spawnBullet(player.pos.add(0,0))
95    })
```

Now if we run it, we can see a rectangle bullet being formed on top of the player whenever we press the spacebar. This is good! However, we need to make some changes to have the bullet in front of the player and give the bullet motion.

3) Let’s make the bullet appear a bit higher up to get it in front of the player by **changing the y coordinate** in the add() function to **-35**

```
93    keyDown('space', () => {
94      spawnBullet(player.pos.add(0, -35))
95    })
96
```

4) Add a **constant for the bullet speed** to our move speeds section

```
78    // set move speeds
79    const MOVE_SPEED = 200
80    const ENEMY_SPEED = 200
81    let CURRENT_SPEED = ENEMY_SPEED
82    const LEVEL_DOWN = 200
83    const BULLET_SPEED = 400
```

5) Use the **onUpdate() function** to give the bullet motion

```
98   // add bullet motion
99   onUpdate('bullet', (b) => {
100    b.move(0, -BULLET_SPEED)
101    if (b.pos.y <0) {
102      destroy(b)
103    }
104  })
```

Destroy Enemies

Now that we can shoot, we want our bullets to destroy enemies upon impact.

1) Use the **onCollide() function** to specify what should happen when a bullet collides with an enemy

```
133  // bullet collides with enemy
134  onCollide('bullet', 'enemy', (b, e) => {
135    shake(2)
136    destroy(b)
137    destroy(e)
138    score.value++
139    score.text = score.value
140  })
```

Here we use the **shake() function** for a camera shake (added fun effect), destroy the enemy and the bullet, and increase the score by 1 for each collision.

Win Conditions

Add a Win Scene

If all the enemies are defeated, we want to have a win scene.

1) Use the **scene() function** to specify what should show on our win screen
   We can add this right below the lose scene, and in this case, we will print the words “You Won!”

```
22  // win scene
23  scene('win', () => {
24    add([
25      text('You Won!'),
26      origin('center'),
27      pos(width() / 2, height() / 2)
28    ])
29  })
```

Create the Win Scenario

If the player destroys all the enemies, we tell the game to go to the win scene,

1) Use the **onUpdate() function** to check for enemies and go to the win scene if there are no enemies left

```
151
152   // all enemies defeated
153   player.onUpdate(() => {
154     if(get('enemy').length == 0) {
155       go('win')
156     }
157   })
158
```

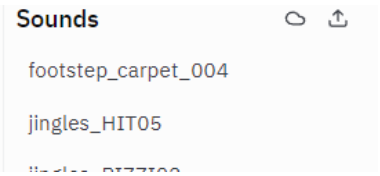
   We have to use the **player.onUpdate** here instead of enemies because when all enemies are defeated, we don’t have the enemy object to check with anymore.

Sound

Uploading Sound to Replit

You can find a lot of free resources online that you can use for sounds. One I would recommend is Kenney free assets. They give you permission to use their sound effects and even characters in any educational, personal, or commercial projects for free! You can download one of their sound packs, unzip the file, listen to the different audio files, and pick the ones you like.

1) In the Sounds section on, click the **upload button** and select the sound file you want to upload. Repeat this step as many times as you want to get all your sound effects in Replit.



Loading Sound into the Game

Loading the sounds into the game uses the same process as loading the sprites into the game. You can hover over the sound and click the 3 dots to Insert the load code. It is good to keep this towards the top of your main.js file for organization. Once your sounds are loaded, change the name in the first set of “ “ to match what you will use that sound for. You do not need to rename the file itself in the side panel or second set of “ ”, but make sure you name it something that will make sense when it is referred to in the code later.

```
5
6   // load assets
7   loadPedit("enemy", "sprites/enemy.pedit");
8   loadPedit("player", "sprites/player.pedit");
9   loadPedit("wall", "sprites/wall.pedit");
10  loadSound("score", "sounds/score.mp3");
11  loadSound("enemy-shot", "sounds/footstep_carpet_004.ogg");
12  loadSound("shoot", "sounds/laserSmall_004.ogg");
13  loadSound("won", "sounds/jingles_PIZZI02.ogg");
14  loadSound("lost", "sounds/jingles_HIT05.ogg");
15
```

Add Sound to Conditions

To have a sound play in the game, we will use the play() function. You can place this anywhere you wish! Below are examples of adding sound to the start of the game, shooting, killing enemies, the win scene, and the lose scene.

Start of the game

```
// add player
const player = add([
  sprite('player'),
  scale(0.5),
  pos(width()/2, height()/1.2),
  origin('center'),
  area(),
  play('score')
])
```

Shooting

```
keyDown('space', () => {
  spawnBullet(player.pos.add(0, -35))
  play('shoot')
})
```

Enemy Shot

```
// bullet collides with enemy
onCollide('bullet', 'enemy', (b, e) => {
  play('enemy-shot')
  shake(2)
  destroy(b)
  destroy(e)
  score.value++
  score.text = score.value
})
```

Win Scene

```
// all enemies defeated
player.onUpdate(() => {
  if(get('enemy').length == 0) {
    play('won')
    go('win')
  }
})
```

Lose Scene

```
// player collides with enemy
player.onCollide('enemy', () => {
  play('lost')
  go('lose')
})

// enemies reach bottom of game board
onUpdate('enemy', (e) => {
  if (e.pos.y >= height() - 70) {
    play('lost')
    go('lose')
  }
})
```

Aesthetics

Change the Background

You can use rgb color values to specify what color you'd like your background to be

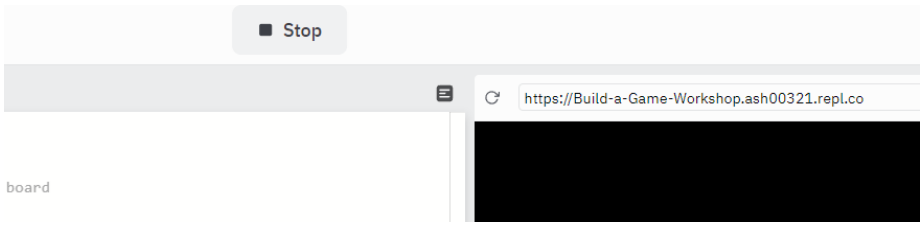
1) On line 4, inside the ( ), you can use background and rgb values

```
2
3   // initialize context
4   kaboom({background: [0, 0, 0]});
5
```

Share Your Game!

If you want others to preview, play, or see the code for, your game:

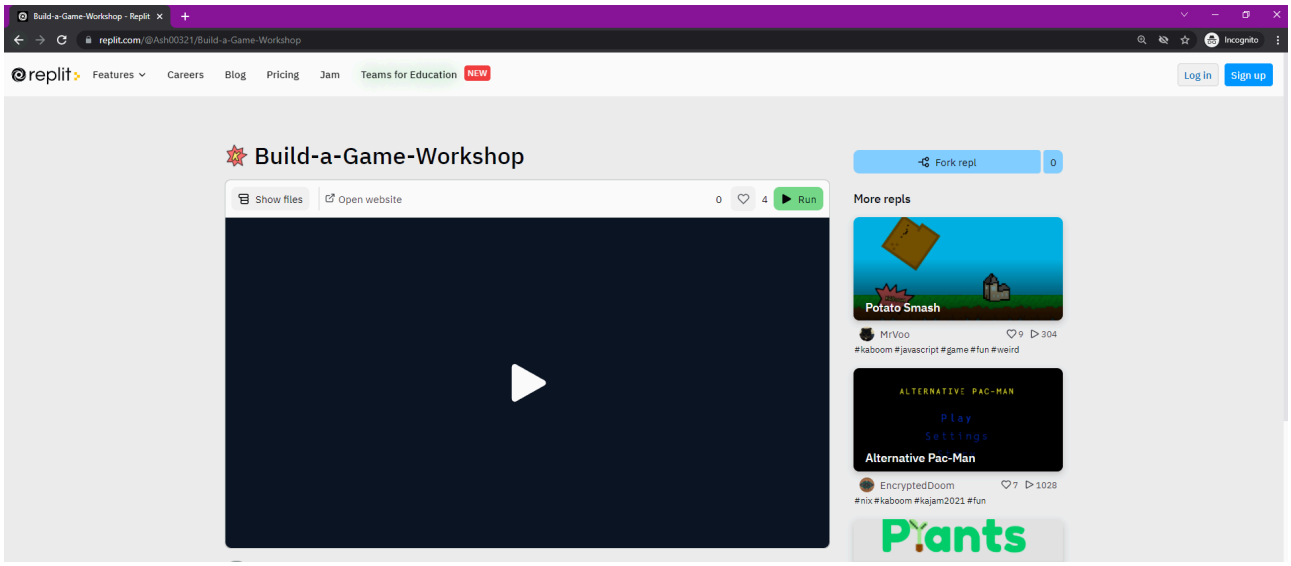
1) Make sure your game is **running**



2) Copy the **link in the top of your screen** (without the #code/main.js)



3) Check this works by **opening an incognito browser** (so you aren't logged in on Replit), and make sure you can play it (CTRL + SHIFT + N for Chrome)



4) If you can't play it, the game probably isn't running. Just go back to your editing view and run the game. You can **leave it running** when you go to close your browser.

Next Level

This was a tutorial on the very basics of making a small-screen Space Invaders game. There are tons of other things you can explore and add into your game! You can make it easier or harder by controlling the number of enemies, the speed of the enemies and the player, etc. Some other suggestions might be:

- Adding new levels
- Have your enemies fire at the player
- Allowing the user to pick the level of difficulty
- Changing bullet spawn rates, speed, etc
- And most importantly, make the game more responsive (current screen size is very specific)

There are also other game tutorials out there, like Mario, Zelda, the Snake Game, and more! Free Code Camp is an awesome resource for all sorts of tutorials, and the one this workshop is based on contains Space Invaders, Mario, and Zelda. Replit also has a Mario tutorial.

Interested in other workshops from Girls Who Code at UNC Charlotte?
Check out our Workshop & Event Resources

Side Notes

Screen View

Inside of your editing window, you may need to drag out the window to see both side walls. You can always click the refresh button in the game preview area to reload the screen with your new dimensions. This can take some playing around to get a good view. See the workshop recording for more on this. You can also check how the dimensions look on the preview screen for others to play by following the Share Your Game! steps. Adjust the wall spacing and sprite sizes as needed.