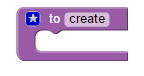
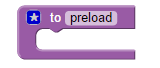
Part 1:

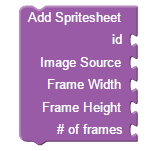
1) Create a game object using the Game > Phaser Game block



2) Create the 3 basic core functions, preload, create and update for the game (these are used in every game you make) using the Functions > do something block. For each function change the "do something" text to preload, create and update



3) Inside the preload function block we want to add the spritesheet image for our player. we use the assets > Add Spritesheet block to do that. For the spritesheet block we want to set the id = "ss" the image source to the spritesheet file game\_starter2/assets/rpg\_guy1.png the frame width = 32 frame height = 48 and # of frame = 16



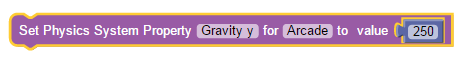
3.6) OPTIONAL: Inside the Create function block set the background color for the game (instead of the black background) using the Game > Set Background Color block



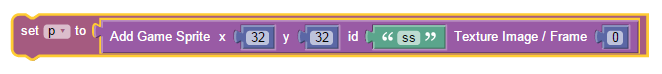
4) We want to add physics to our game so we need to use the Physics > Start Physics block and choose Arcade

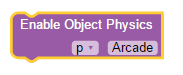


4.5) Set a game wide gravity value for the Y axis. We do this using a Physics > Set Physics System Property block and set the Gravity Y dropdown to a value of 250

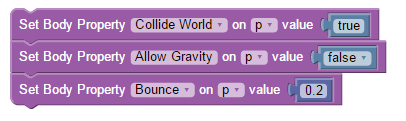


5) We want to place our player sprite in the world. In the create function, add a new variable using the Variables > Set item to block change the item dropdown to New Variable and change the name to p. in the value socket of the block insert a Assets > Add Game Sprite block. Set the values for x and y = 32 (each) set id = ss (what we called the spritesheet in the preload function) and set frame = 0. The frame value is for telling the game which one of our 16 spritesheet frames to show when the sprite is added to the screen

6) Now we want to add physics to the player. This will make sure the player doesn’t walk through walls or platforms, be pulled by gravity and get hit by enemy weapons. Use the Physics > Enable Object Physics and select the p from the item dropdown



7) Still in the create function, set different Physics properties for the player. We really just need 3. Collide World, Allow Gravity and Bounce. We do this with 3 of the same blocks Physics > Set Body Property. We set Collide World to True, set Allow Gravity to False and Set Bounce to 0.2



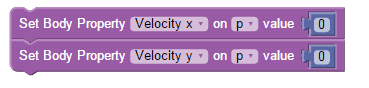
8) For our game we want to follow the player. To do that we set the game camera to follow the player around. We use the Game > Camera Follow block and set the item dropdown to p (for the player)



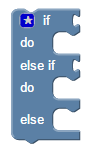
9) To move the player around we want to use the Arrow Keys on the keyboard. We tell the game to use the arrow keys by using the Input > Enable Cursor Keys block, but we need to assign that block to a variable using the Variables > Set item to block . We insert the Input > Enable Cursor Keys block into the variable value and create a new variable from the item dropdown and setting the name to cursors.



9.5) When we start checking the player movement, we want to set the velocity for the player to 0. This keeps the player from moving in strange directions when we don’t want it to. To do that we add 2 Physics > Set Body Property blocks to the beginning of the update function block. We set both the Velocity X and Velocity Y properties to 0



10) Now we need to make the player move around based on which arrow key is pressed. We do this by checking which key is pressed. Checks are done using If Else ElseIf blocks. Since the player will move left, right, up and down there are 2 things we need to do, 1) check if the player is going up or down and 2) if they are going left and right. In this game, these 2 things DO NOT happen together. You either go up, go down, go left or go right. There is no diagonal movement. This can be done with 2 If, ElseIf Else blocks. First drag a Logic > If Do block onto the work area, click the little star in the corner of the block. Inside that little box that looks like a speech bubble, drag the Elseif block into the If block, then under the Elseif block drag the Else block. Close the speech bubble by clicking the star again. To save some work, duplicate the If block you just created and drag it into the Else area of the first block you created.



11) In the If we want to check for any cursor key being pressed. When the If ElseIf Else blocks are nested, you will see there are 4 connectors. We can use each of these connectors for each arrow key. (Up, Down, Left and Right) So If block lets us check for something. To check which arrow key is pressed we use the Input > Check If Arrow Key block. When you add all the check arrow blocks, into the If, snap the nested If blocks into the update function block.



12) Now that we know that an arrow is pressed, we can make the player react. For this game we want to move the player around the screen. We will move the player using the Player Object's Velocity property. Velocity is the speed of how fast the player will move. To change the velocity of the Player we use the Physics > Set Body Property block.



The player will move up, down, left and right. To move Up and Down we want to change the Velocity Y Property of the player. To Move Left and Right, we will change the Velocity X.

In the If Elseif Else, block you added to the update function, for each arrow key we want to add a set Velocity block for the matching direction.

Our game screen has X and Y values, When you move Left, the X values get smaller, when You move right, the X values get larger. So, to move Left we need to Set the Velocity X to a negative number and when we move Right we change it to a Positive number. When the player moves Down the Y values get larger and when the player moves Up The Y values get smaller. So to move Up we need to change the Velocity Y to a negative Number and when the player moves Down we change the Velocity Y to a positive number.

12.5) When the player moves up we need to change Velocity Y to -175. When the player moves down we need to change Velocity Y to 175. Then to move Left we change Velocity X to -175 and to move Right we change Velocity X to 175.

13) The player is moving around the screen but it would be better if we animate the character in the direction its facing. The spritesheet we loaded contains the 4 animations want to use, Facing left, Facing Right, Facing Up and Facing Down. To use these animations we need to add each one to our create function using the Animation > Add animation block.



For each animation we need to add a list of frames (from the spritesheet) to use for the animation. We can create a list using the Lists > Create list with block



We insert into the list block the numbers of the frames for each animation. Each Direction Animation will have 4 frames. Left frame numbers are: 4,5,6,7; Right frame numbers are: 8,9,10,11 ; Up frame numbers are: 12,13,14,15; and Down frame numbers are: 0,1,2,3

In addition, to the 4 direction animations, we want to create an animation with a single frame to use as a standing still animation for the character. The Add Animation block for the Idle (standing still animation) looks like:



14) Now that we have added our animations we can use them when one of the arrow keys are pressed. So in the update function, where we defined our big If ElseIf Else block, we want to add each direction animation right before the Set Velocity X and Set Velocity Y blocks. To start an animation, we use the Aniimation > Play animation block



we change the id to the corresponding direction up, down, left, right.

At the bottom of the If ElseIf Else block (the last else section) we want to add the play animation block for Idle. The last else, is used when there no arrow key being pressed.

Now our character moves all around the screen with full animation.

Part 2:

Its great that our character moves around but it would be better if we had a nice world level to have our player explore. Lets add one.

15) In our preload function lets add a level map to our game by using the Assets > Load Tilemap block

We want to give the map an Id of level. These are called Tilemaps and can pick 1 of 4 gamemap files (gamemap,gamemap2,gamemap3,gamemap4) for our level.

16) For our Tilemap to work, we need to add an image file to our game that contains all the tiles used to draw our level. This image is called a Tileset. To add the Tileset image to our game we use the Assets > Load Image block.



We give our Tileset image the id tiles. This image contains a set of tiles used to generate the map level. Each tile in this set is 32 pixels wide by 32 pixels tall.

17) Now we want to create a new tilemap using the Add Tilemap block. To do that we want to assign a new variable named map.



When we create a new tilemap we specify the level map we loaded before and set the width and height of the tiles to 32.

18) The map file defines how to draw the screen. However, to actually see the map you need to tell it which tiles to use. This is done with the Game Map > Add Tileset From block.



19) When you create a tilemap and add a tileset, you are simply displaying the map on the screen. To make that map part of the game, you need to create a Map Layer. Creating a map layer adds game objects which the player can interact with. To create a map layer, we set a new variable, layer, using the Game Map > Create Map Layer block



20) Our new layer may be bigger than the game screen size we set at the beginning of our program, so we want to expand our world to fit the map layer. We do this by using the Game Map > Resize World to Layer block.



21) If we save and run the game, we will see our new map appear in our game screen. Now we have our character and our scene.

22) You may notice that as you move around the world, you are actually able to walk through walls. So our map is more like a background. Lets make the map a bit more exciting and have the player collide with the walls of the map to give them a challenge to navigate around the obstacles.

23) To add collision to the map, we need to tell our game which tiles are walkable and which tiles are wall tiles. When we setup tiles earlier in the program, we used a tileset. In the tileset, each tile has an ID number, these ID numbers are used to represent a Wall or a Walkable tile. Right now, we only need to tell the game which tile IDs are walls. We do that using the Game Map > Set Tile Collision for map block. There are 2 versions of this block, one version allows you to set 1 map ID



And the second version lets you set a range of IDs (tile IDs between 2 numbers



Depending on your tilemap the collision tile IDs will be different.

24) Even though you have specified collision tiles, your game still does not know that you want the player to collide with wall tiles. To tell the game to make the player collide with the we we use the Physics > Add Collision between block in the Update function and specify Arcade and set the collision between the player object (p) and the map layer (layer). Since we do not need to do anything when the player collides into the wall, we do not need to change Executing null.



Now our player is able to move around the world and crash into wall.

