Lesson Plan

Title	SET Camp
Subject	Video Game Camp
Author	Christopher Foley
Grade level	Middle
Time duration	2-5 day
Overview	Students will create a video game
Objective	Shooter Game (saucers vs aliens) using Drag and Drop. Secondary objective might be platform game.
Materials	https://www.yoyogames.com/downloads/TutorialResources
Activities and procedures	See attached.
Conclusions	Students can create a video game
Extra credit	Knowledge itself is enough credit

DAY	ACTIVITIES
MONDAY	Camp Structure, Fundamental Concepts of GameMaker, Movement, Shooting, Enemies, Tiles and Views
TUESDAY	Drawing Text, Drawing objects, sounds, spawning objects, title screen, maintaining score, high score screens, paths, image manipulation
WEDNESDAY	Develop rudimentary platform game. Guide students through their design and implementation of their game. Students should determine what they enjoy in other one and two player games versus what they do not like.
THURSDY	Guide Students in the design and implementation of their games. Students should base their games on what they enjoy in other games.
FRIDAY	Complete Game, Demonstrate Game for other camps, recreation breaks

DnD Saucers vs Alien Spawn

- SET VideoGame Design Camp Strustures
  - Following yoyogames.com tutorials
  - Discuss Lab rules
  - Discuss computer use rules
  - Discuss copyright laws
  - No phones
  - Remote login and Z: drive
    - https://tsg.eng.fau.edu/software/remote-desktop-access/
    - Login: fau_ab…b (a-first letter of first name, b….b – last name)
    - Password: GOOWLS2018 or GOOWLS2018!
    - Install VMWare Horizon from tsg.eng.fau.edu page
  - Unzip TutorialResources.zip from yoyogames site
    - Locate where they are stored (TutorialResources directory)
  - Decision was made to use Drag and Drop (DnD) to speed up new developers
  - Gamemaker 2.0 allows code changes to DnD
- Fundamental concepts
  - Sprites – how things appear
    - Create spr_player
    - Create spr_enemy
    - Create spr_bullet
  - Objects – interact with objects and make things happen
    - Create obj_player with spr_player
    - Create obj_enemy with spr_enemy
    - Create obj_bullet with spr_bullet
  - Rooms – where things happen
    - Create rm_game with obj_player
- Movement
  - Add movement to obj_player
    - Discuss orientation of room with respect to (x,y) (Graphic not cartesian)
    - Step function called 30 times per second VERY CRITICAL TO REMEMBER THIS
    - **Keyboard** events for arrow keys and/or WASD keys
    - Test the key interaction by moving 4 pixels per event

- "point" to an object (code or set variable)
  - Inside step event add code or set variable (this will appear to track real-time)
    - image_angle = point_direction(x, y, mouse_x, mouse_y)
  - Show player object rotating with mouse
  - Discuss zero angle as horizontal to right ("east") with angles counter clockwise:

    0 – horizontal to right

    90- up

    180 – horizontal to left

    270 – down

- Later we will create a wrap function to wrap from edge to edge. **Note** wrap function from menu requires direction AND speed to be set which implies we do not control speed and direction in step function. Gamemaker tutorials state that this requires a non-trivial amount of CPU power so dong it in code is easier.

- Shooting
  - Bullets
    - Inside player step event look for left mouse down to generate
    - Create within obj_player step at x,y with direction equal to direction of player use player image_angle or simply use point_direction(x, y, mouse_x, mouse_y) ; use speed 8
    - Fire some bullets and observe what happens
    - Look at bullets for out of room/boundary condition destroy when outside room
      - Inside obj_bullet look for boundary event and destroy self
    - Observe bullets how they overlap and create cooldown variable of 15 (15 steps between shots)
      - Player create event cooldown = 15
      - Player step subtract 1 (set variable -1 **relative**) from cooldown
      - Player step check for cooldown equals 0 then fire
      - TEST firing sequence (15 cooldown = 2 per second)
- Enemies
  - Discuss collision events enemy / bullet
  - Bullet collision with obj_enemy destroy self otherwise the bullet will continue through object and off screen, if you have health points (below) one undestroyed bullet can kill an enemy.
  - healthpoints – subtract 1 for each collision, when 0 destroy
  - Image angle in step image_angle=point_direction (x, y, obj_player.x, obj_player.y); this orients the enemy angle to the player object. This will not work if multiple obj_player objects

are on the game board. If there are other slave players then use one obj_player and the rest obj_slave_player.
    - Move toward player using move towards point (obj_player.x, obj_player.y) since we have one player this works

- Tiles and Views
    - Create background(s)
    - Use background as a tileset
    - Bg_tileset from tutorial
    - Change width 128x128
    - Using tiles/tileset – tile background
    - Expand room and retile
    - Camers/views
        - View following obj_player
        - Discuss border size
- Drawing Text: Score
    - Draw event
    - When overriding draw do a draw self (unless no sprite is present)
    - Create a font
    - Draw score (system variable score then local score)
        - First position x=room_width, y=16
        - It doesn't follow player outside room
        - View_xview+view_wview/2 and view_yview+16 (leaves a 16 pixel gap from top in center of view)
    - Draw healthbar for enemy health variable vs user defined.
    - `draw_healthbar(x1, y1, x2, y2, amount, backcol, mincol, maxcol, direction, showback, showborder);`
    - draw_healthbar(x-57, y-65, x+57, y-60, health_points*20, c_black, c_red, c_green, true, true)
- Sounds
    - In camp computer lab, headphones are needed to listen to sound.
    - Create sounds for games
    - Need to get my pew pew sound. It's an annoying sound.
    - Discuss the irritation level of hearing sounds repeatedly. Gamemaker 2.0 allows parameters to be modified randomly.
    - Teacher computer takes a few seconds to begin playing sounds. May not be worth the effort.
- Title Screen/Levels

- Title screen is created as a room
- Create title screen with title object
- Discuss persistent object that can monitor keyboard for pause and exit.
  - Persistent objects remain for all rooms
  - Use this to monitor and define keys that are consistent for ALL rooms!!!!!
- Rooms may be switched by "next_room" which follows the room definition list or goto_room.
- The first room executed is at the top of the Room list.

- Spawning
  - Create obj_spawner or obj_evil_overlord
    - Middle school students prefer to call it obj_evil_overlord
  - Creates enemies every 2 seconds (use alarm) to create obj_enemy_spawn
  - Obj_enemy_spawn initially displayed at 0.10 size, then each step increases by 0.02
  - When size is 1.0 or greater fix at 1.0 and change to obj_enemy
  - Discuss execute events (CREATE EVENT) for change instance
  - Discuss spawn counter
- High Score Screen
  - High score table initialize in title screen (Create obj_initialize)[i]
  - When player dies or all enemies, spawn enemies and spawn count are 0 (all enemies are dead, none are expected to be created)
  - Check score is greater than highscore_value[10] then use get_string[ii]
  - Add_highscore(name, value) discuss that it inserts name/value at appropriate location in table. Since we compared to highscore_value[10] we know we made the top 10.
  - Write table using function[iii]
  - Draw high score table (set font, set colour then draw)[iv]
    - Table is 473 x 350 pixels in size
      - Subtracting from width and height of room then dividing remainder by 2 to get offset hx, hy (upper left corner of high score table)
- Paths
  - Show how paths work, define points or circuit.
  - Paths are open or closed
  - Reverse at end or restart?
    - Reverse at closed path reverses path
    - Restart at open, jumps to open
  - Add a path on high score screen
  - Campers can create a dummy player

        - Follows path
        - Does not die
        - Automatically shoots at a single enemy on screen
        - Spawns a new enemy when one is killed
            - Step function counts enemy and spawn enemy if both are zero – spawn
    - Nested if statements needed. Use this as animation behind High Score display.
- Timelines
    - Enemies can die a more spectacular death!
    - Create obj_dying_enemy
    - Create enemy death sprites.
    - Use timeline to change sprite every ½ second
    - Start enemy with flash
    - Change enemy death sequence to obj_dying sprite
- Image Manipuation
    - Transparent backgrounds
    - Gamemaker manipulation
    - GIMP ?

Platform Game

- Follow Shawn Spaulding's Tutorial
- Sprites
  - Player sprite, use multiple sprites for jumping, walking, standing
  - Wall sprite – basic block to represent a wall
- Objects
  - Player object – does all the work in code. It can be done DnD.
    - Player code is as follows: Code Design comments are bullet items

```
// get player input

key_left=keyboard_check(vk_left);

key_right=keyboard_check(vk_right);

key_jump=keyboard_check_pressed(vk_space);
```

- The keyboard_check methods return 0,1 so we store them in variables. The left direction is negative and the right direction is positive therefore we subtract left from right. Move will then equal the direction as a **signed** value. If both left and right are pressed, move is zero.

```
//calculate movement

var move = key_right - key_left;
```

- walksp is the number of pixels we are going left/right, vsp is vertical plus gravity which is down/positive.

```
hsp = move*walksp;

vsp = vsp + grv;
```

- If we are not under a wall, we allow a jump

```
if (place_meeting(x,y+1,obj_wall) && key_jump)

{

   vsp = -7;       // jump up

}
```

- We then loop checking for horizontal and vertical motion **one pixel at a time.** We use sign() and ceil() to get either 1 or -1. This is necessary because adding fractional pixels can cause the player to get stuck in an object, if care is not taken.

```
// check for horizontal collisions
if (place_meeting(x+hsp,y,obj_wall))
{
   while (!place_meeting(x+sign(hsp),y,obj_wall))
   {
      x += sign(hsp);
   }
   hsp = 0;
}
x += hsp;

// check for vertical collisions
if (place_meeting(x,y+vsp,obj_wall))
{
   vsp = ceil(vsp);
   while (!place_meeting(x,y+sign(vsp),obj_wall))
   {
      y += sign(vsp);
   }
   vsp = 0;
}
y += vsp;
```

- Animated gif or different sprites are used when jumping and running. When there is no motion, the sprite is normal.

```
// animation
if (!place_meeting(x,y+1,obj_wall))
{
   sprite_index = spr_jumping;
   image_speed = 0;
   if (sign(vsp) > 0)
   {
      image_index = 0;
   }
   else
   {
      image_index = 1;
   };

}
else
{
   image_speed = 15;
   if (hsp ==0)
   {
      sprite_index = spr_basic;
   }
   else
   {
      sprite_index = spr_walking;
   }
```

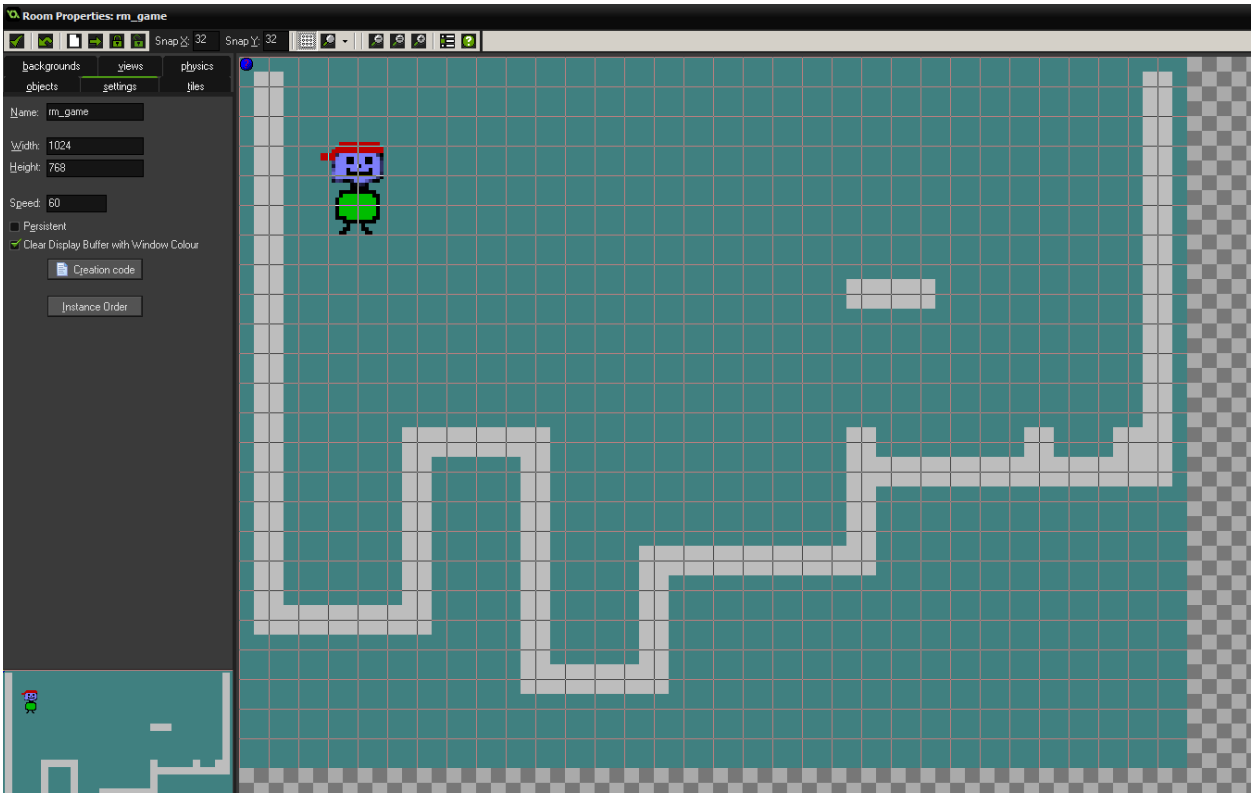
}

Image_xcale = -1 reflects the image over its y axis. This allows the sprite to face left (-1) or right (1)

if (move < 0) image_xscale = -1 else image_xscale = 1;

- Wall object – has no actions – its' a wall
- Control object – persistent but no sprite, this initializes the values at the start

- Multiple game rooms may be used
- Our example only needed one room (switching between rooms was demonstrated in shooter game)
- The following room was defined for the above code:



[i] Obj_initialize – code to read high score table.

```
highscore_clear();
if (file_exists("asteroids.ini"))
{
  ini_open("asteroids.ini");
  for (var i=1; i<=10; i++)
```

```
   {
    name = ini_read_string("HighScore", "name" + string(i), "");
    value = ini_read_real("HighScore", "value" + string(i), 0);
    highscore_add(name, value);
   }
  ini_close();
}
```

[ii] Get_string for player name:

```
global.Name = get_string("Congratulations, you have a high score# please enter your name: ", "<anonymous>");
highscore_add(global.Name, score);
```

[iii] Write High score table:

```
ini_open("asteroids.ini");
for (var i=1; i<=10; i++)
{
   ini_write_string("HighScore", "name" + string(i), highscore_name(i));
   ini_write_string("HighScore", "value" + string(i), string(highscore_value(i)));
}
ini_close();
```

[iv] Draw High Score:

```
hx=view_xview+(view_wview-473)/3;
hy=view_yview+(view_hview-350)/3;
draw_highscore(hx, hy, hx+473, hy+473);
```