Winter 2025 Mid-Semester Exam - Question Packet

By Austin Yarger - University of Michigan (ayarger@umich.edu)

Glossary

Helpful Engine Lua API Functions
Image.DrawUI(string image_name, int x, int y) – Draw an image at (x,y) in screen space.
Audio.Play(int channel, string clip_name, bool does_loop)
Input.GetKey(key) -- Returns true if the key is currently down on this frame.
Input.GetKeyDown(key) – Returns true if the key was pressed down this frame.
Actor.Find(string name) – Returns a single actor reference by name.
Actor.FindAll(string name) – Returns all actors of a given name.
Scene.Load(scene_name) -- Loads a scene of a certain name.
actor_ref:GetComponent(component_type_name) – Get reference to a component by type.
Image.Draw(image_name, int x, int y) – Draws image in scene space (x,y) (**note :** no floats)

Helpful Common Lua Functions
math.abs(number) -- absolute value
math.max (a, b) -- Get max number of a and b. Opposite of math.min(a, b)
table.insert (my_table, thing_to_add) – Adds a thing to a table at the very end.
table.remove(my_table, index_to_remove) -- Removes the item at “index_to_remove”
#table_name -- Get the number of items in a table.

Helpful Lua Techniques
for index, value in ipairs(table) do <logic_here> end – iterating through a table w/ index.
for i = #my_table, 1, -1 do <logic_here> end – Iterating backwards through a table.
some_number % 2 == 0 – using modulo to check if some number is even.

```
if <condition> then
-- Do Stuff
elseif <confiditon> then
-- Do Stuff
else
-- Do Stuff
end
```

May

Magna / “Mags”

Helpful C++ / Pseudocode Techniques

```
// Retrieving a property from a component
// component below is a LuaRef.
luabridge::LuaRef property_of_component = component["enabled"];
if (property_of_component.cast<bool>() == true)
        // Do something fun
```

LuaRef my_function = component_luaref[“my_function”]; // Get a function on lua component
my_function(component_luaref); // call it and send in a reference to “self” (member function).

std_data_structure.clear() // clear common std data structures such as vector, map, etc.

for (LuaRef & lua_thing : lua_refs) { } // for-each iteration.

Data type sizes (for the purposes of this exam)

char	1
int	4
<any pointer type>	8
std::string	24

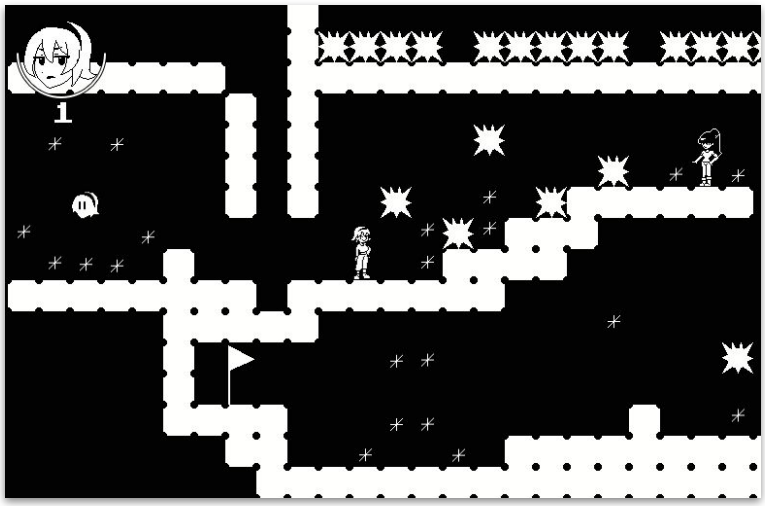
17

__/10

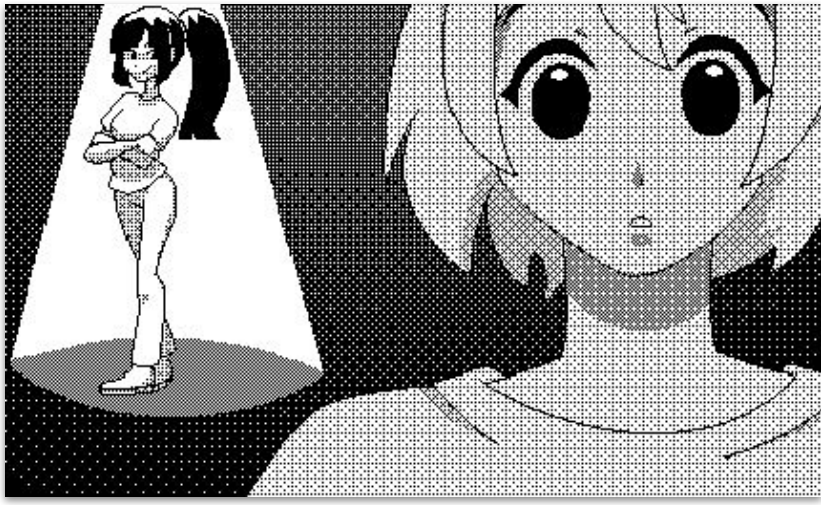
Frenemies Forever

Games benefit tremendously from storytelling. Game designers need our help and technology to establish narrative, characters, theming, and relationships.

Cutscenes bring images, text, and audio together to tell a story. In this problem, you will write a couple cutscenes for *May and Mags*, the game on the course website, *eecs498.com*.



May and Mags, a retro-style 2D platformer in which players help May overcome competitors in a wild athletics competition.



teamup1.png

May's sister Magna, the game's primary antagonist, appears during a cutscene. Space bar progresses the images.

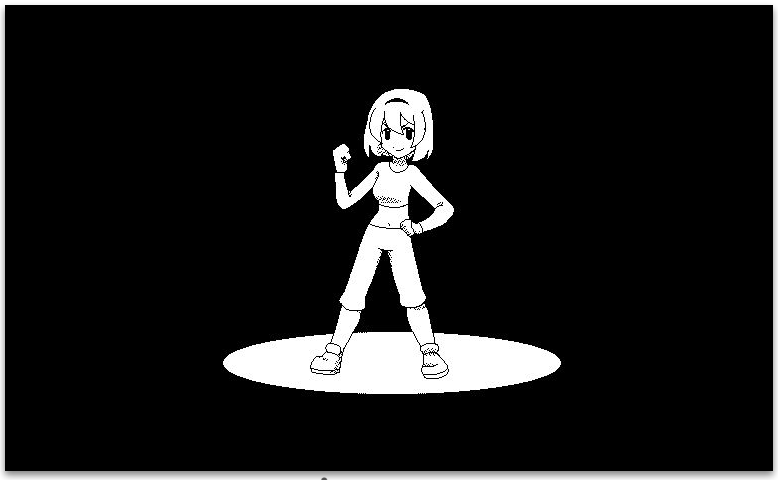
teamup2.png

The sisters, normally fierce rivals, form a temporary alliance.

teamup3.png

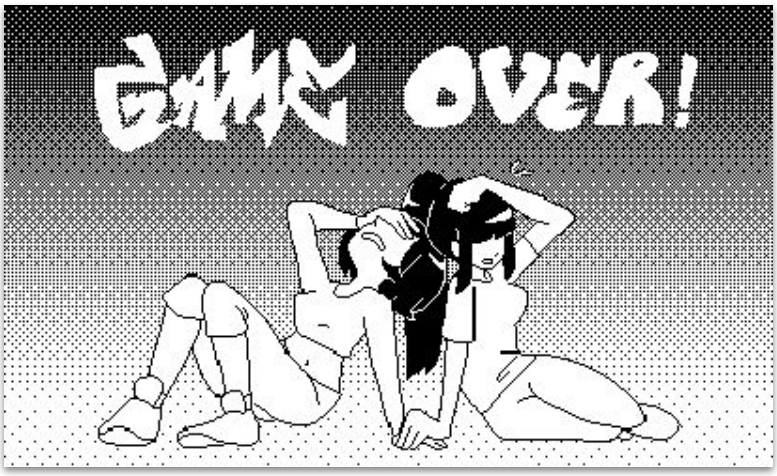
In this boss fight, both characters get their crack at the boss– a cool biker dude.

Lua and Composition



victory.png

The sisters are victorious, but the mood is ominous. Magna departs suddenly. They'll face each other in the final battle.



gameover.png

If both May and Magna run out of lives, they are both eliminated from the tournament.

Objective

Write a "CutsceneManager" Lua component to draw cutscene-related UI in response to the "May" and "Magna" actors (the player characters) and the player's pressing of space bar.
Note : If a requirement is ambiguous or absent = your choice (different ideas may be valid).
Note : Use the glossary at the front of this exam for a reminder of Lua functions / technique.

Requirements

☐ Write your Lua component in the corresponding box on your answer document.
☐ All lua code is valid / syntactically correct.
☐ Component makes use of Lua lifecycle function(s) from lecture / homeworks.

GameOver State

- The "May" and "Magna" actors both have a "Status" component, which houses a "lives" int.
  - ☐ If, at any point, both lives counters are <= 0, DrawUI gameover.png at (0,0)
  - ☐ This image should draw atop everything else / occlude everything else in this problem. (Do not worry about gameplay in this problem).

Victory State

- At any moment, some number of actors with the name "enemy" exist.
  - ☐ If, on any frame, the number of enemy actors is == 0, DrawUI victory.png at (0,0)
  - ☐ This image should occlude everything besides the game over screen.

Cutscene State

- If the game has not concluded in victory or defeat yet, show a cutscene.
  - ☐ DrawUI teamup1.png at 0,0. Move onto subsequent images with every spacebar press ("space")"
  - ☐ When you have exhausted all images, draw nothing (allows gameplay to proceed) (Do not worry about gameplay this problem. Another dev is handling it.)

18

__/10

“Leave it to me!”
A great game contains interesting, impactful decisions, and the decision of “who shall I play as” is a classic of the medium. From replay value to representation, a charming character select screen starts a game off right. Let’s create one.

Example : Super Mario Bros. 2 / USA
A theater-themed Character Select Manager.
The manager “Instantiates” available characters, and the characters handle rendering and animation themselves.

Each character handles its own rendering. Upon being “informed” they have been selected, a pose occurs. Scene changes to gameplay shortly thereafter.

Objective
Write two Lua components (**CharacterSelectManager** and **Character**) that work together to bring a character select menu, and the characters themselves, to life.

Hint : Think about the centralized responsibility of the manager (to spawn / load the characters in and help the player choose between them) and the decentralized responsibility of the individual characters (each character actor rendering and animating itself and receiving information from the manager so it may do so).

Note : If a requirement is ambiguous or absent = your choice (different ideas may be valid).
Note : Use the glossary at the front of this exam for a reminder of Lua functions / technique.

Please see requirements on the next page.

Requirements for CharacterSelectManager component

☐ Write your Lua component in the box on the following pages. Use valid Lua code.
☐ Component makes use of Lua lifecycle function(s) from lecture / homeworks.

☐ A global table called “character_datas” exists, and contains a table for each playable character. Every table contains data describing a unique character–
- ☐ “idle” : the name of a character’s idle sprite (that they use while standing around).
- ☐ “pose” : the name of a character’s posing sprite (that they use after being selected).
- ☐ “clip” : the name of the audio voice clip played upon selecting the character.

☐ For each table in the character_datas table, Instantiate an actor from template “Character”. This Character template comes with a “Character” component (written by you).
- ☐ “Inform” the Character component of the position it should render at
  X = index_in_character_datas_table * 100, Y = 300
- ☐ “Inform” the Character component of its “idle”, “pose”, and “clip” information.

☐ The manager keeps track of the “hovered char” and allows the player to change and select it.
- ☐ Pressing “right” increases an index (representing the current “hovered” character).
- ☐ Pressing “left” decreases this index.
- ☐ This index is “wrapped” to prevent index-out-of-range errors / nils
  (pressing “right” on final character wraps to the beginning character and vice-versa)
- ☐ Upon pressing “space”, inform the “hovered character” it has been selected.

Requirements for Character component

☐ Write your Lua component in the box on the following pages. Use valid Lua code.
☐ Component makes use of Lua lifecycle function(s) from lecture / homeworks.

- Recall that the manager component above will instantiate multiple actors with this component, as well as provide certain pieces of information (“idle”, “pose”, “clip”, “x”, “y”) to help it do its job of representing a single character.

☐ DrawUI the “idle” image (whatever image came from the manager) on every frame at the x and y location provided from the manager unless it has been “selected” by the manager.
☐ Upon being selected by the manager, immediately play the “clip” audio file (provided by the manager) on channel 1 with no looping. Do so exactly once (and not every frame).
☐ For approximately 120 frames (~2 seconds) after being selected, DrawUI the “pose” image at the provided x and y position.
☐ On approximately the 120th frame since being selected, load the “gameplay” scene.

19

__/ 9

A game designer has used your engine to make a game, and sends you the source as a keepsake. You've officially "made it" as an engine developer.

Upon inspecting the "resources" folder that represents the game, you find–

resources/images


circle.png


victory.png


square.png


ko.png

resources/actor_templates/player.template

```
{
  "name": "player",
  "components": {
    "1t": {
      "type":"Transform",
      "x":1
    },
    "2kc": {
      "type":"KeyboardControls"
    },
    "3sr": {
      "type":"SpriteRenderer",
      "image": "circle"
    }
  }
}
```

resources/actor_templates/enemy.template

```
{
  "components": {
    "1t": {
      "type":"Transform",
      "x":2, "y":-2
    },
    "2ai": {
      "type":"EnemyAI"
    },
    "3sr": {
      "type":"SpriteRenderer",
      "image": "square"
    },
  }
}
```

resources/component_types/KeyboardControls.lua

```
KeyboardControls = {
  OnStart = function(self)
    self.t = self.actor:GetComponent("Transform")
    score = 0
  end,

  OnUpdate = function(self)
    if Input.GetKey("up") then self.t.y = self.t.y - 1 end

    if Input.GetKey("down") then self.t.y = self.t.y + 1 end

    if Input.GetKey("right") then self.t.x = self.t.x + 1 end

    if Input.GetKey("left") then self.t.x = self.t.x - 1 end

  end
}
```

resources/component_types/Transform.lua

```
Transform = {
  x = 0,
  y = 0
}
```

Engine Architecture and Lifecycle Functions

resources/component_types/EnemyAI.lua

```
EnemyAI = {
  OnStart = function(self)
    self.t = self.actor:GetComponent("Transform")
    local player = Actor.Find("player")
    self.pt = player:GetComponent("Transform")
    self.sr = self.actor:GetComponent(“SpriteRenderer”)
  end,

  OnUpdate = function(self)
    if self.t.x == pt.x and self.t.y == pt.y then
      self.sr.image = "ko"
      score = score + 1
      if score >= 2 then
            Actor.Find(“player”):GetComponent(“SpriteRenderer”).image = “victory”
      end
      self.enabled = false
      return
    end

    if Application.GetFrame() % 2 == 0 then
      self:RunAI()
    end
  end,

  RunAI = function(self)
      local dx = self.t.x - self.pt.x
      local dy = self.t.y - self.pt.y

      if dy == 0 then self.t.y = self.t.y - 1
      elseif dy > 0 then -- Do nothing
      elseif dx >= 0 then self.t.x = self.t.x + 1
      elseif dx < 0 then self.t.x = self.t.x - 1
      end

      self.t.x = math.min(7, math.max(0, self.t.x))
      self.t.y = math.min(0, math.max(-3, self.t.y))
  end

}
```

resources/component_types/SpriteRenderer.lua

```
SpriteRenderer = {

  image = "",

  OnUpdate = function(self)
    local t = self.actor:GetComponent("Transform")
    Image.Draw(self.image, t.x, t.y)
  end

}
```

resources/scenes/level1.scene

```
{
  "actors": [
     {
          "name": "baddieSouth",
          "template": "enemy",
          "components": {
               "1t": {
                    "x": 2,
                    "y": 0
               }
          }
     },
     {
          "name": "baddieNorth",
          "template": "enemy",
     },
     {
          "name": "baddieWest",
          "template": "enemy",
          "components": {
               "1t": {
                    "x": 0
               }
          }
     },
```

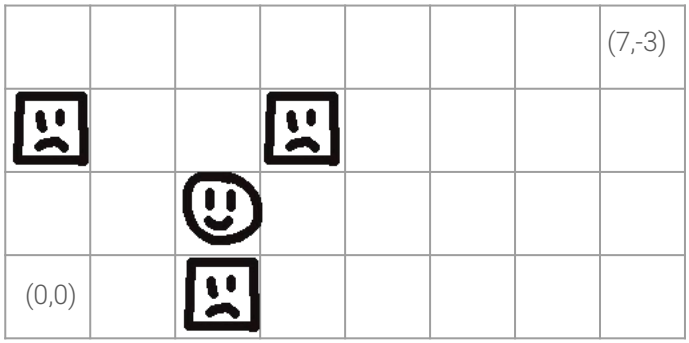
resources/scenes/level1.scene **(continued)**

```
     {
          "name": "angry bill",
          "template": "player",
          "components": {
               "1t": {
                    "x": 1,
                    "y": -1
               }
          }
     }
  ]
}
```

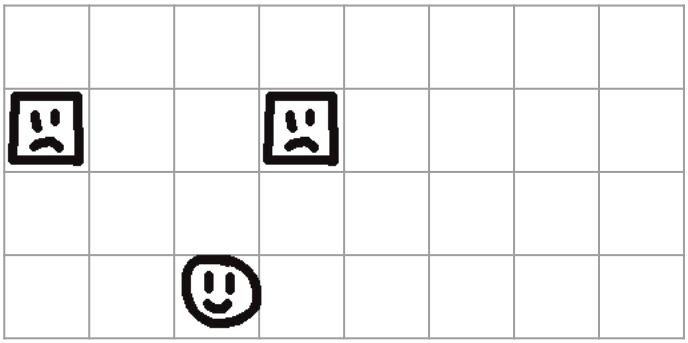
Objective

Draw the first 10 frames of the game. Take note of the input following the frame, if any.

- The first two frames are provided for you below. The world is 8x4 in dimension.
- The "initial_scene" is "level1.scene". Bottom-left cell is 0,0 while top-right is 7,-3
- Recall that these "renders" are the last thing to occur in a frame (after all logic).
- **Note :** Actors occlude each other if they stand in the same cell. No transparency.



Frame #0 (input coming early frame : **right**)



Frame #1 (input coming early frame : **down**)

20

__/10

“Lifecycle” Functions : Connect() and Disconnect()

Game designers have begun using your engine to create impressive 2D games.

Sadly, they have thus far all been single-player. Game designers want the ability to connect to a network, process network data using functions in their components, then disconnect. As an example, one might write a component that receives coordinates from the network, then moves its actor there to show the presence of online player characters (see the example below).

While not being lifecycle functions themselves, Connect() and Disconnect() effectively allow the user to create their own new Lifecycle function– one that receives network data when it arrives for a given actor. See below–

```
ChangePositionToRepresentOnlinePlayer = {

      OnStart = function(self)
            -- Register a component (and a function) to receive network data.
            -- Key is used to identify the component the function belongs to.
            -- Multiple functions may be registered.
            self.actor:Connect(self.key, self.OnNetworkDataMove)
            self.actor:Connect(self.key, self.OnNetworkDataLog)
      end,

       -- Called by engine when network data arrives for this actor.
       -- “data” is a string.
      OnNetworkDataMove = function(self, data)
            local transform = self.actor:GetComponent(“Transform”)

            -- split the data string (“x,y”) into x y coordinates
            local x, y = data:match("(%d+),(%d+)")

            -- move our transform to these coordinates.
            self.transform.x = tonumber(x)
            self.transform.y = tonumber(y)
      end,

      OnNetworkDataLog = function(self, data)
            print(data)
      end

      OnDestroy = function(self)
            -- This actor no longer needs network data. Unregister everything.
            self.actor::Disconnect()
      end
}
```

Your objective
In order to bring these “lifecycle” functions to life and get network data delivered to the correct lua components (and lua functions registered on those components…)

Look over the files and functions for this problem in the answer packet. Then…

1) Decide on one instance / non-static data structure to add to Actor.h
2) Write the Connect() function so that your data structure tracks registrations.
3) Write the Disconnect() function so that your data structure gets “cleared out”.
4) Write the OnNetworkDataArrived() function to deliver data to registered Lua functions.

Assume LuaRef is copyable / easy to pass around without side effects.

Hints
Check the glossary’s C++ / LuaBridge section.
Does this remind you of a certain design pattern? Perhaps one with a fun name?
Components have certain useful data injected into them, such as “key”, “actor”, etc.