SPLIT SCREEN

Francesc Teruel Rodríguez | Personal Research

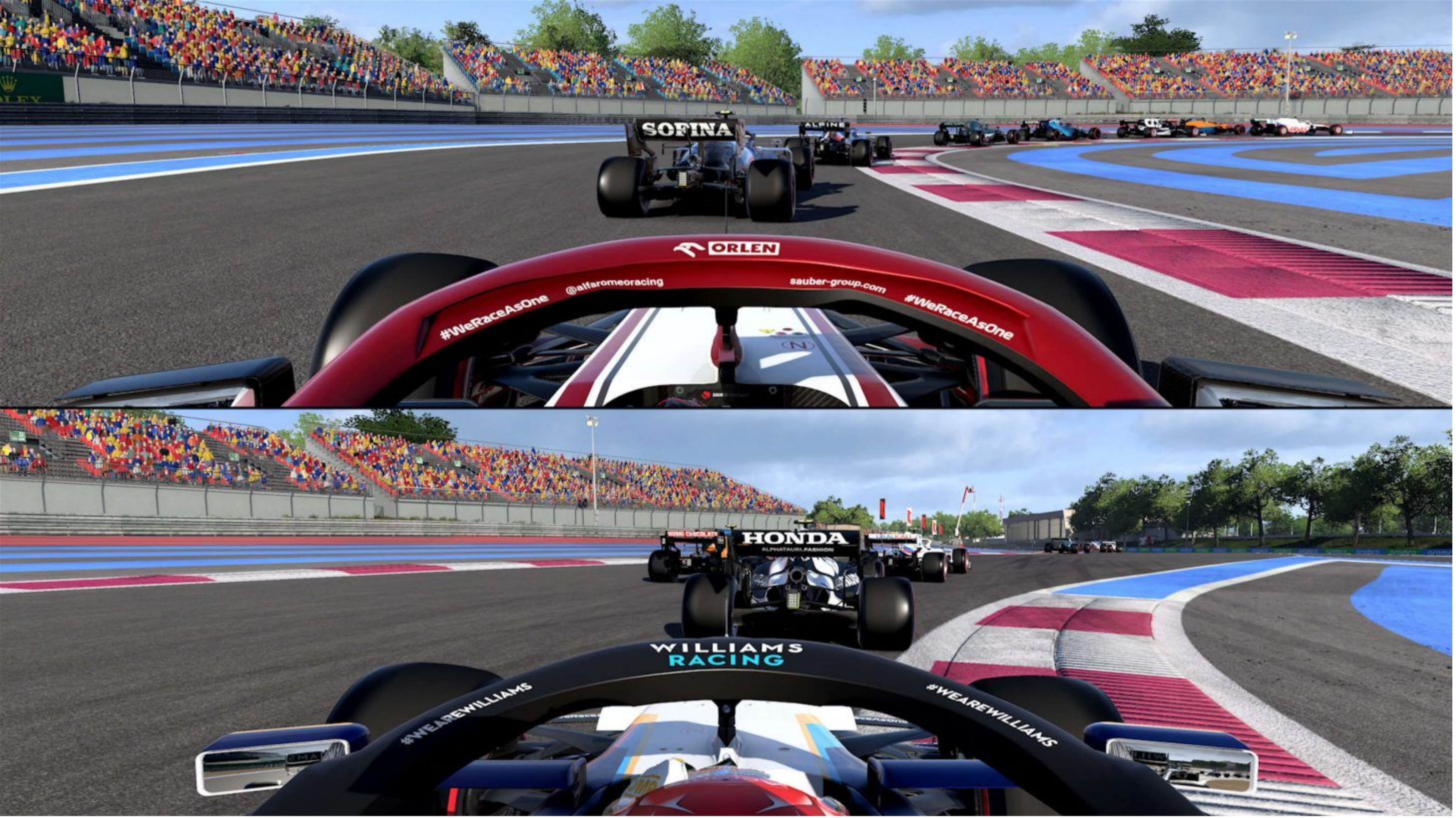
INDEX

1. Introduction
2. Current state
3. Selected approach
4. Split Screen Handout
5. Handout Solution
6. Possible improvements
7. Conclusions
8. References

1. INTRODUCTION

- **What is a Split Screen and what is it used for?**

The Split Screen is an **audiovisual output display** where the screen has been divided into **two or more exactly equal areas** so that players can explore different areas simultaneously without having to be close to each other.



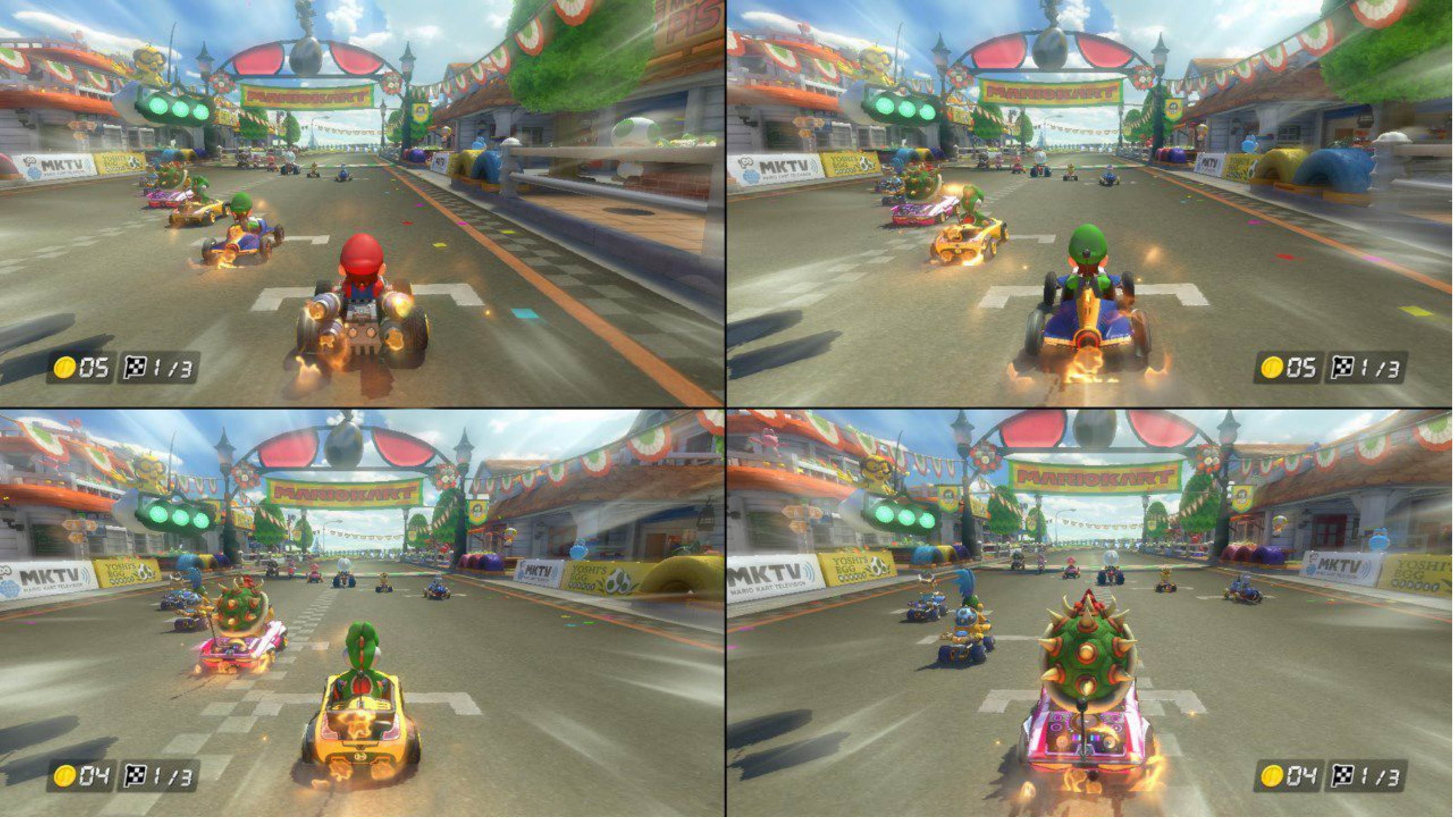
F1 2021 Split Screen

1. INTRODUCTION

- **Why is it important to have a Split Screen?**

The Split Screen feature is commonly used in **non-networked multiplayer video games**, also known as *couch co-op*, and allows multiple people to play on a single device.

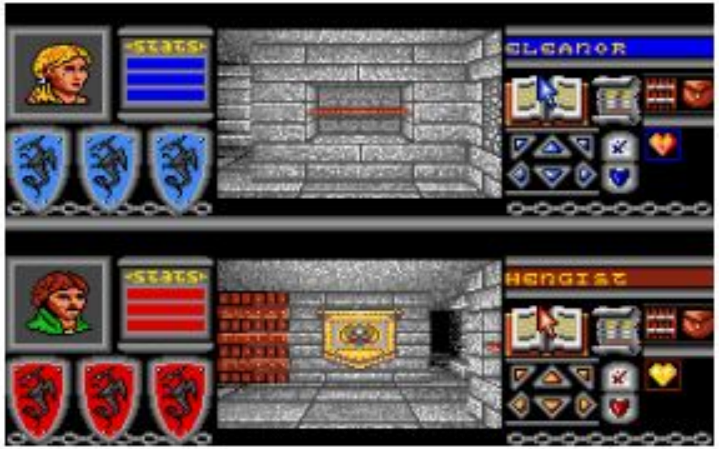
For the context of our **project**, we may find it useful to implement a Split Screen.



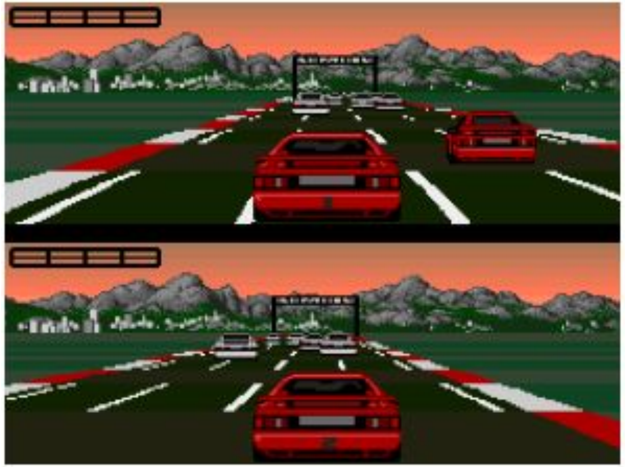
Mario Kart 8 Deluxe Split Screen

1. INTRODUCTION

- **Context and History**



➢ Bloodwych (1989)



➢ Lotus Esprit Turbo Challenge (1990)

1. INTRODUCTION

- **Context and History**

➢ Lemming (1991)

➢ GoldenEye 007 (1997)

1. INTRODUCTION

- **Context and History**

➢ Halo: Combat Evolved (2001)

➢ Call of Duty 2 (2005)

1. INTRODUCTION

- **Current references**

Rocket League Split Screen

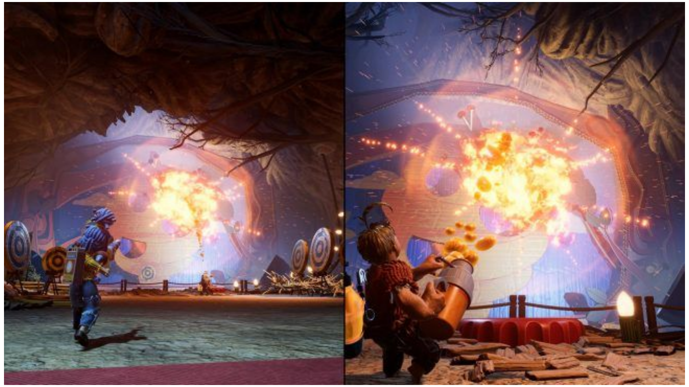
It Takes Two Split Screen

2. CURRENT STATE

- **Technical evolution of the Split Screen**

From ... ... To

Drag Race (1977)

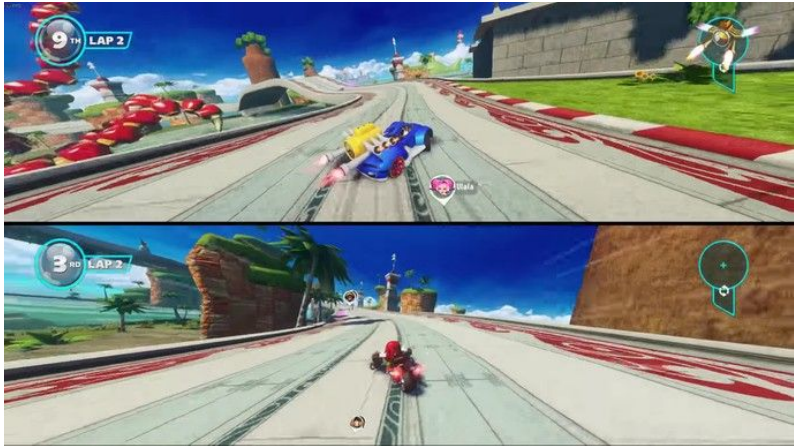


It Takes Two (2021)

2. CURRENT STATE

- **Current techniques in the industry**

➢ Standard Split Screen



Sonic & All-Stars Racing Transformed Collection (2013)

2. CURRENT STATE

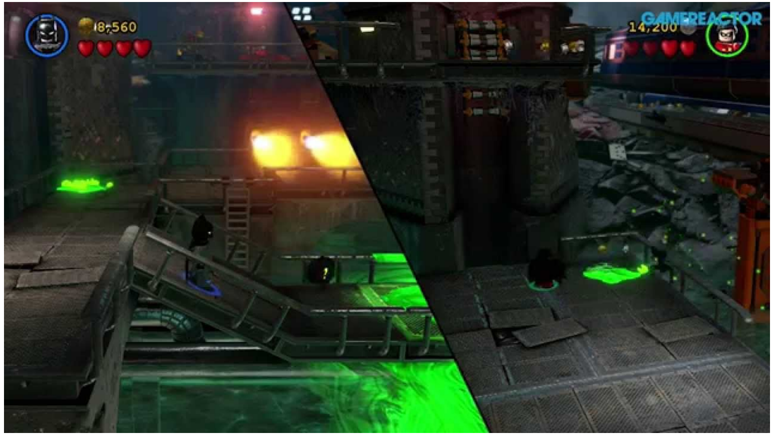
- **Current techniques in the industry**

➢ Voronoi Split Screen

2. CURRENT STATE

- **Current techniques in the industry**

➢ Voronoi Split Screen



Lego Batman 3: Beyond Gotham (2014)

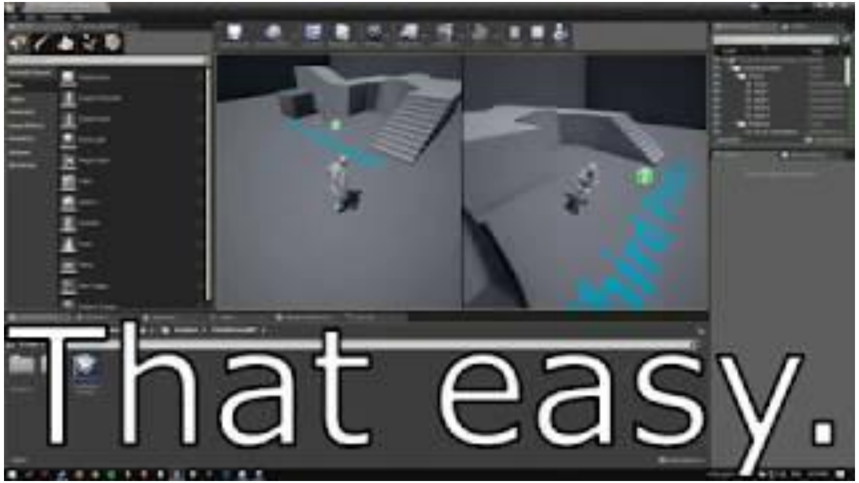
2. CURRENT STATE

- **Split Screen in other game engines**

Split Screen in Unity

2. CURRENT STATE

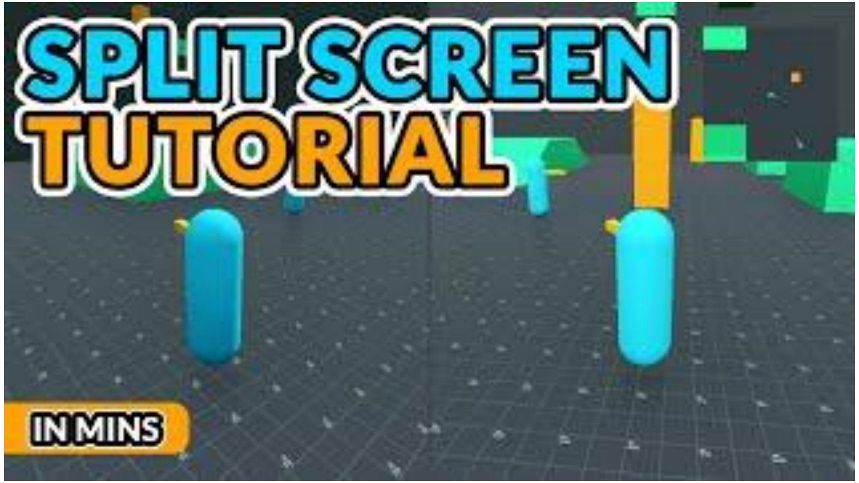
- **Split Screen in other game engines**



Split Screen in Unreal Engine 4

2. CURRENT STATE

- **Split Screen in other game engines**



Split Screen in Godot Engine 3.4

3. SELECTED APPROACH

- ***Split Screen in SDL2*** → Our objective.
- **Basic concepts**
  - ➢ Window
  - ➢ Render
  - ➢ Renderer
  - ➢ Camera
  - ➢ Viewport

3. SELECTED APPROACH

- **Basic concepts**

  ➢ **Window:** A separate viewing area on a computer display screen as part of a GUI.

  ➢ **Render:** The process that turns the code you write on an application into something interactive.

  ➢ **Renderer:** The piece of code that turns code instructions into an interactive rendering context.

3. SELECTED APPROACH

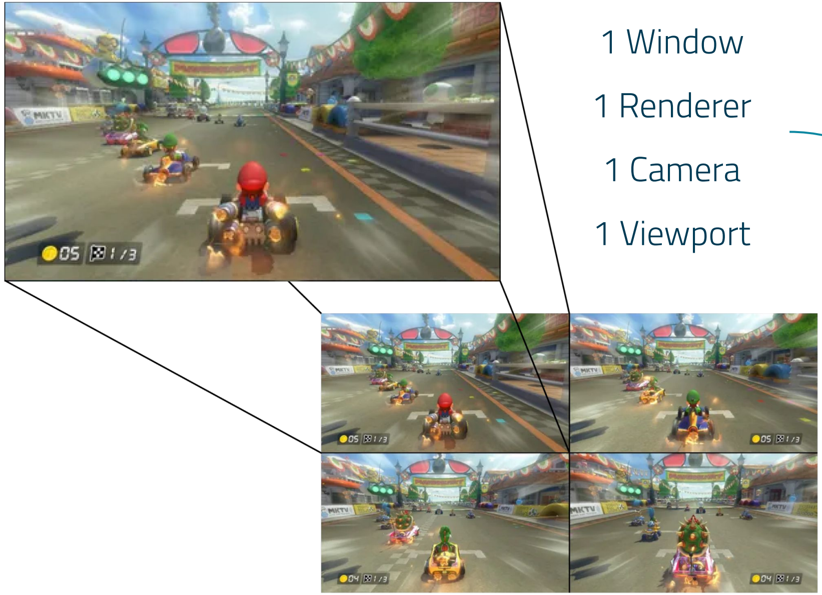
- **Basic concepts**
  - ➢ **Camera:** Designates the point of view that the players will have presented on their screens.
  - ➢ **Viewport:** Is a region of the screen used to display a portion of the total image to be shown.

3. SELECTED APPROACH

- **¿Renderer vs Camera vs Viewport?**

3. SELECTED APPROACH

- **How could we *theoretically* code a Split Screen?**



1 Window

1 Renderer

1 Camera

1 Viewport

x4

1 Window

4 Renderers

4 Cameras

4 Viewports

3. SELECTED APPROACH

- **SDL2 is evil → We have to look for another way**

➢ **We can't create multiple renderers.**

➢ **We can't create multiple cameras.**

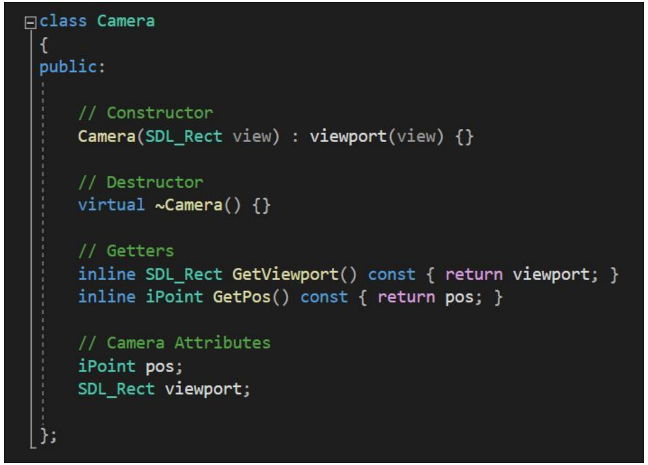
➢ **We can't create multiple viewports.**

(At least, not directly)

3. SELECTED APPROACH

- **Then, how can we program a Split Screen *in SDL2*?**

➢ *New features:*



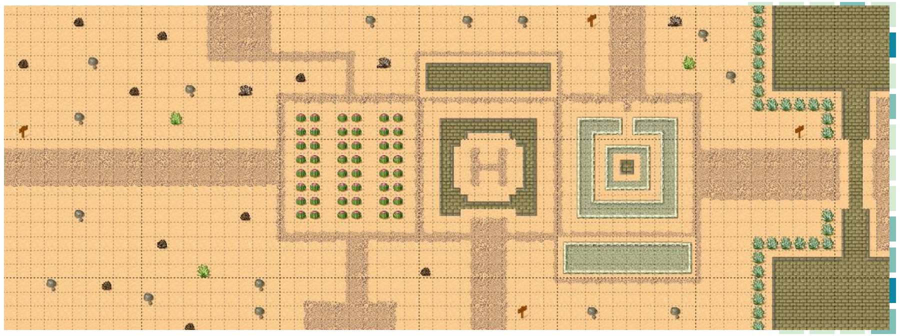
```
class Camera
{
public:

    // Constructor
    Camera(SDL_Rect view) : viewport(view) {}

    // Destructor
    virtual ~Camera() {}

    // Getters
    inline SDL_Rect GetViewport() const { return viewport; }
    inline iPoint GetPos() const { return pos; }

    // Camera Attributes
    iPoint pos;
    SDL_Rect viewport;

};
```

player1.png

player2.png

player3.png

player4.png



3. SELECTED APPROACH

- **Then, how can we program a Split Screen *in SDL2*?**

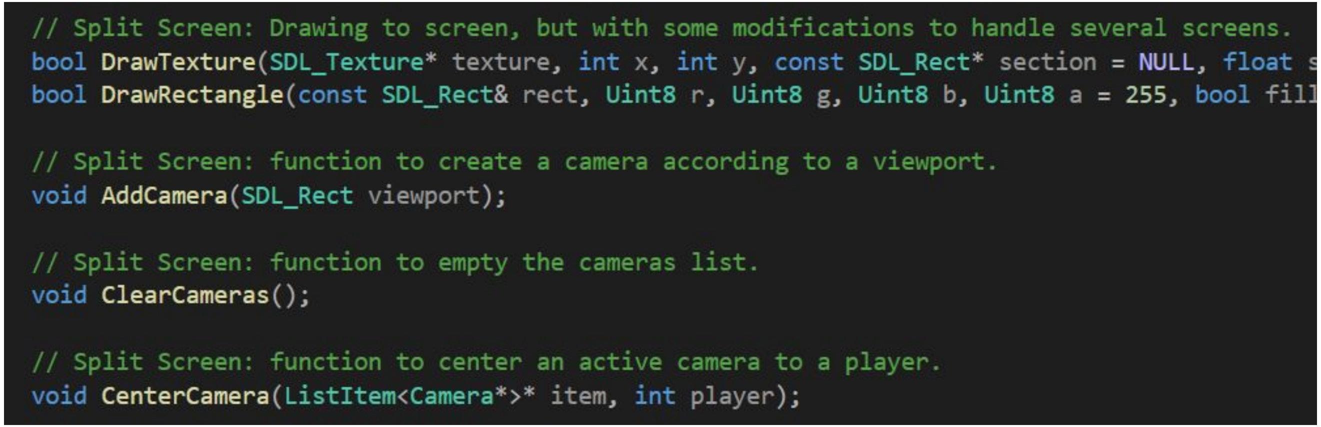
➢ *Affected modules:*

- **Camera** → The camera attribute is now a class.
- **Player** → Multiple players with different input.
- **Scene** → The scene will split in **n** screens.
- **Render** → Manage multiple viewports and cameras.

3. SELECTED APPROACH

- **Then, how can we program a Split Screen *in SDL2*?**

➢ *New functions:*

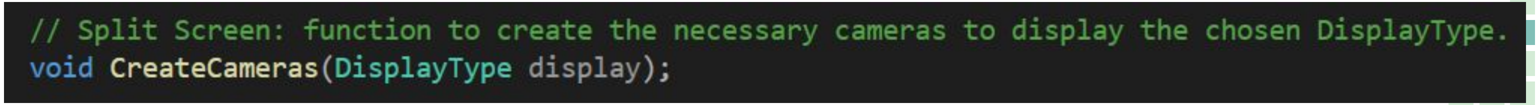


```
// Split Screen: Drawing to screen, but with some modifications to handle several screens.
bool DrawTexture(SDL_Texture* texture, int x, int y, const SDL_Rect* section = NULL, float s
bool DrawRectangle(const SDL_Rect& rect, Uint8 r, Uint8 g, Uint8 b, Uint8 a = 255, bool fill

// Split Screen: function to create a camera according to a viewport.
void AddCamera(SDL_Rect viewport);

// Split Screen: function to empty the cameras list.
void ClearCameras();

// Split Screen: function to center an active camera to a player.
void CenterCamera(ListItem<Camera*>* item, int player);
```

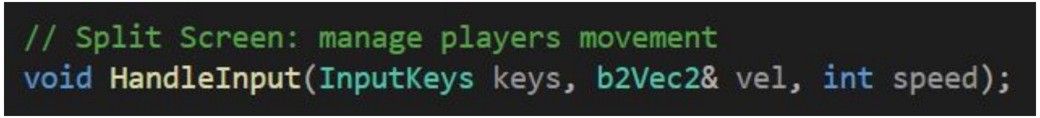
3. SELECTED APPROACH

- **Then, how can we program a Split Screen *in SDL2*?**

➢ *New functions:*



```
// Split Screen: function to create the necessary cameras to display the chosen DisplayType.
void CreateCameras(DisplayType display);
```



```
// Split Screen: manage players movement
void HandleInput(InputKeys keys, b2Vec2& vel, int speed);
```

3. SELECTED APPROACH

- **Then, how can we program a Split Screen *in SDL2*?**

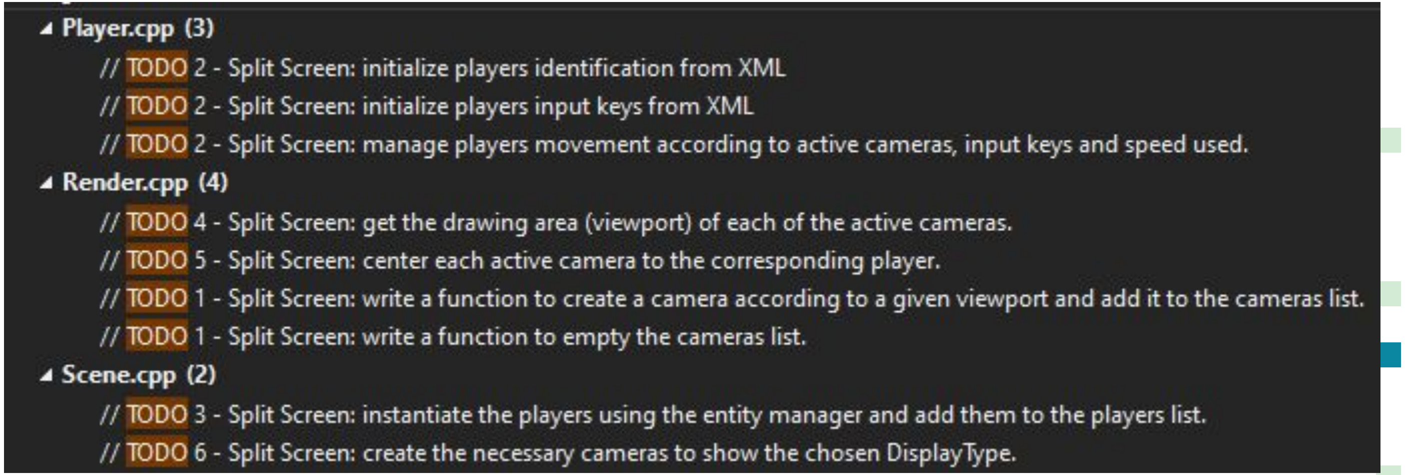
➢ *Steps to follow:*

- ○ 1. Create the cameras and add them to a list.
- ○ 2. Relate the different players with the cameras.
- ○ 3. Assign and center the cameras to the players.
- ○ 4. Finally, display the desired cameras.

3. SELECTED APPROACH

- **Then, how can we program a Split Screen *in SDL2*?**

➢ *Steps to follow:*



```
◢ Player.cpp (3)
      // TODO 2 - Split Screen: initialize players identification from XML
      // TODO 2 - Split Screen: initialize players input keys from XML
      // TODO 2 - Split Screen: manage players movement according to active cameras, input keys and speed used.
◢ Render.cpp (4)
      // TODO 4 - Split Screen: get the drawing area (viewport) of each of the active cameras.
      // TODO 5 - Split Screen: center each active camera to the corresponding player.
      // TODO 1 - Split Screen: write a function to create a camera according to a given viewport and add it to the cameras list.
      // TODO 1 - Split Screen: write a function to empty the cameras list.
◢ Scene.cpp (2)
      // TODO 3 - Split Screen: instantiate the players using the entity manager and add them to the players list.
      // TODO 6 - Split Screen: create the necessary cameras to show the chosen DisplayType.
```

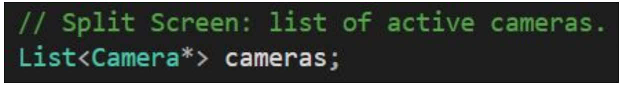
NOW
LETS GET SOME WORK DONE
makeameme.org

4. SPLIT SCREEN HANDOUT

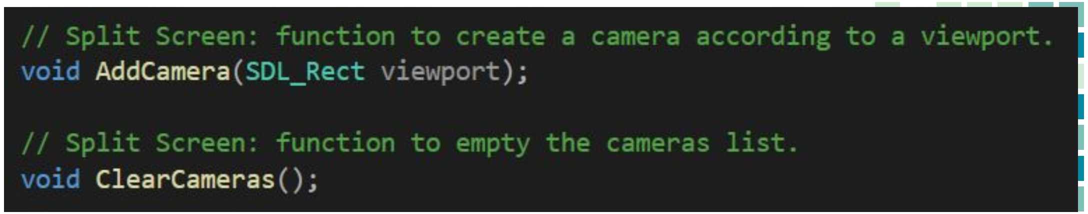
💀 **Download Handout** 💀

TODO 1

- **[Render.cpp] "Create cameras according to a viewport"**
  - ➢ First create a Camera* with the viewport defined.
  - ➢ Then add the camera to the cameras list.
  - ➢ Don't forget to clear the cameras list.



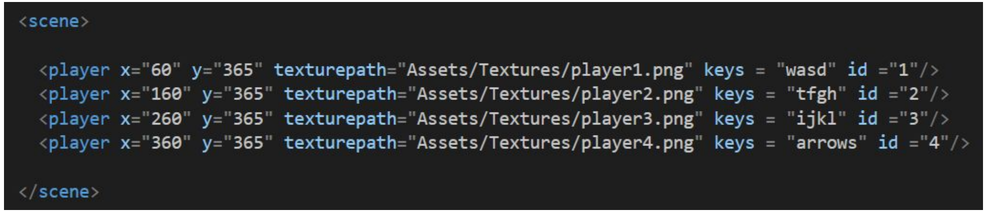
```
// Split Screen: list of active cameras.
List<Camera*> cameras;
```



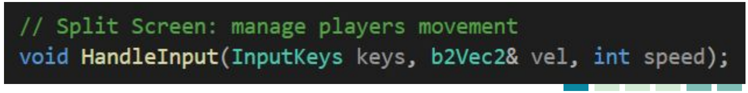
```
// Split Screen: function to create a camera according to a viewport.
void AddCamera(SDL_Rect viewport);

// Split Screen: function to empty the cameras list.
void ClearCameras();
```

TODO 2

- **[Player.cpp] “Manage players movement and camera”**
  - ➢ Read the parameters from config.xml
  - ➢ Then use the function *HandleInput* to manage the camera-player-input relation



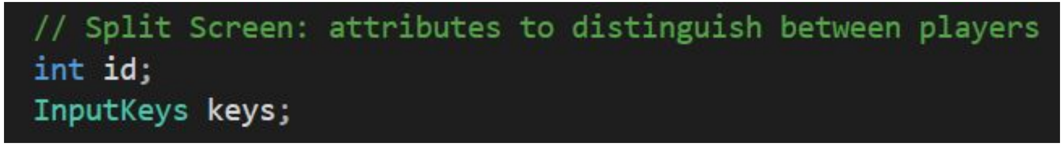
```
<scene>

  <player x="60" y="365" texturepath="Assets/Textures/player1.png" keys = "wasd" id ="1"/>
  <player x="160" y="365" texturepath="Assets/Textures/player2.png" keys = "tfgh" id ="2"/>
  <player x="260" y="365" texturepath="Assets/Textures/player3.png" keys = "ijkl" id ="3"/>
  <player x="360" y="365" texturepath="Assets/Textures/player4.png" keys = "arrows" id ="4"/>

</scene>
```



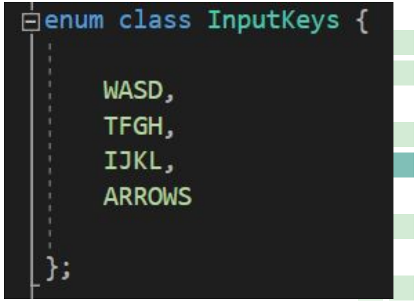
```
// Split Screen: manage players movement
void HandleInput(InputKeys keys, b2Vec2& vel, int speed);
```

TODO 2

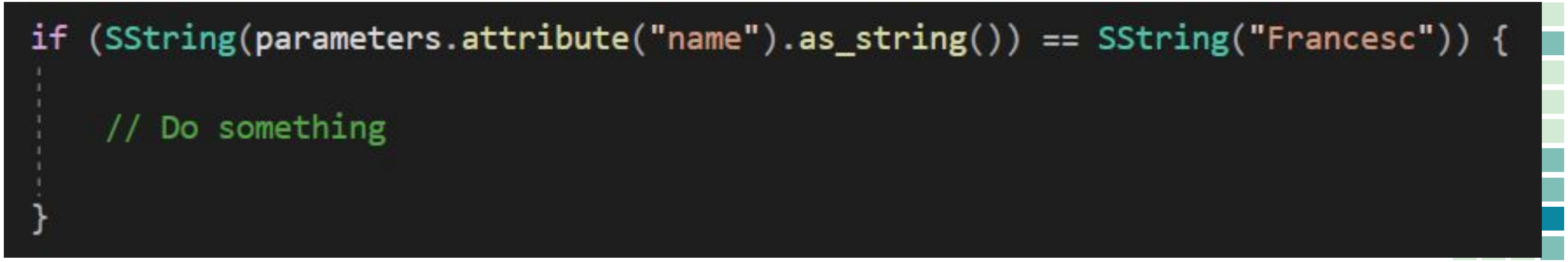
- **Additional information**



```
// Split Screen: attributes to distinguish between players
int id;
InputKeys keys;
```



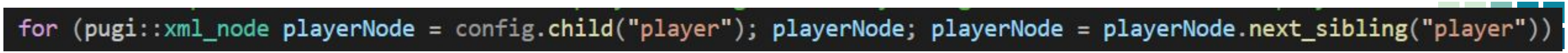
```
enum class InputKeys {

    WASD,
    TFGH,
    IJKL,
    ARROWS

};
```



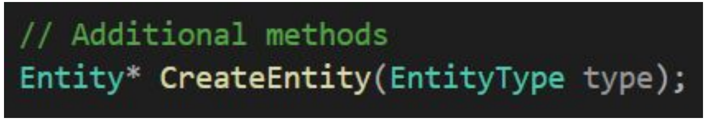
```
if (SString(parameters.attribute("name").as_string()) == SString("Francesc")) {

    // Do something

}
```

TODO 3

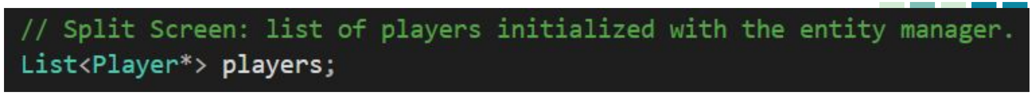
- **[Scene.cpp] “Instantiate the players in the scene”**
  - ➢ Read the config.xml and retrieve all the player nodes.
  - ➢ Then you have to *CreateEntity* of each player.
  - ➢ Don’t forget to add them to the players list.



```
for (pugi::xml_node playerNode = config.child("player"); playerNode; playerNode = playerNode.next_sibling("player"))
```



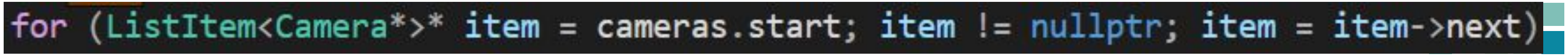
```
// Additional methods
Entity* CreateEntity(EntityType type);
```



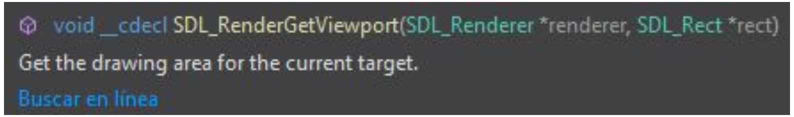
```
// Split Screen: list of players initialized with the entity manager.
List<Player*> players;
```

TODO 4

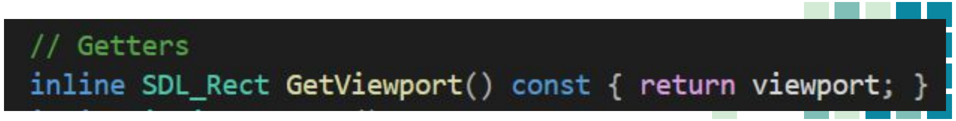
- **[Render.cpp] “Get the viewport of each active camera”**
  - ➢ You need a loop to go through the list of cameras
  - ➢ For each camera, state renderer and viewport
  - ➢ You can retrieve the viewport of the cameras



```
for (ListItem<Camera*>* item = cameras.start; item != nullptr; item = item->next)
```



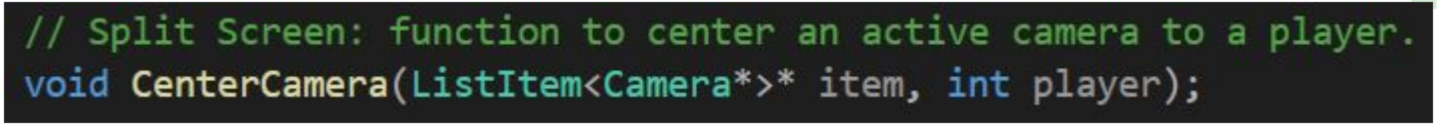
```
void __cdecl SDL_RenderGetViewport(SDL_Renderer *renderer, SDL_Rect *rect)
Get the drawing area for the current target.
Buscar en línea
```



```
// Getters
inline SDL_Rect GetViewport() const { return viewport; }
```

TODO 5

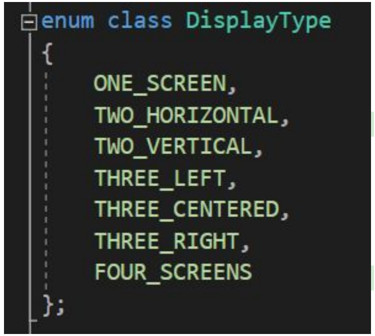
- **[Render.cpp] “Center each active camera to a player”**
  - ➢ Very similar loop to TODO 4
  - ➢ But now the loop has two iterators
  - ➢ You have to increment it after each iteration



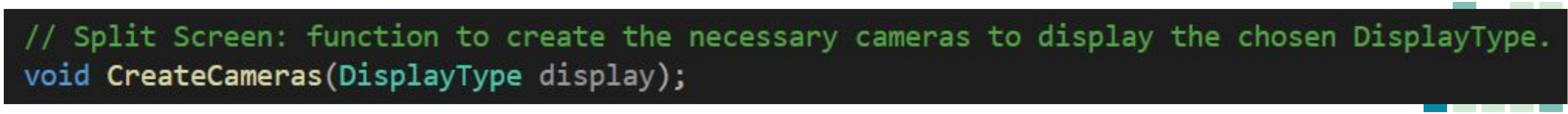
```
// Split Screen: function to center an active camera to a player.
void CenterCamera(ListItem<Camera*>* item, int player);
```

TODO 6

- **[Scene.cpp] "Create all the necessary cameras"**
  - ➢ Check the function *CreateCameras*
  - ➢ Check the enum class *DisplayType*



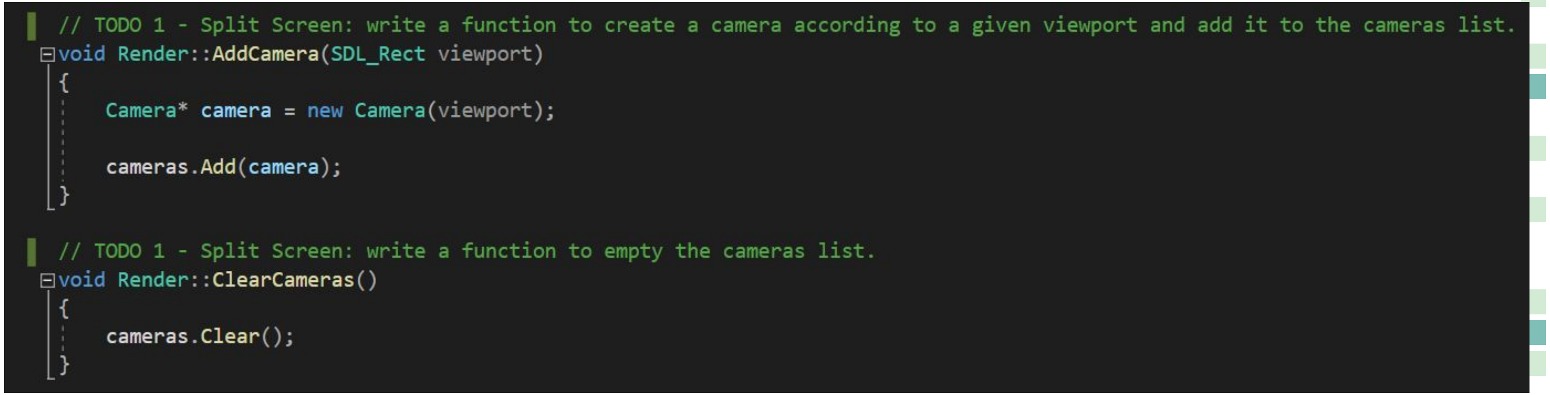
```
enum class DisplayType
{
    ONE_SCREEN,
    TWO_HORIZONTAL,
    TWO_VERTICAL,
    THREE_LEFT,
    THREE_CENTERED,
    THREE_RIGHT,
    FOUR_SCREENS
};
```



```
// Split Screen: function to create the necessary cameras to display the chosen DisplayType.
void CreateCameras(DisplayType display);
```

5. HANDOUT SOLUTION – TODO 1

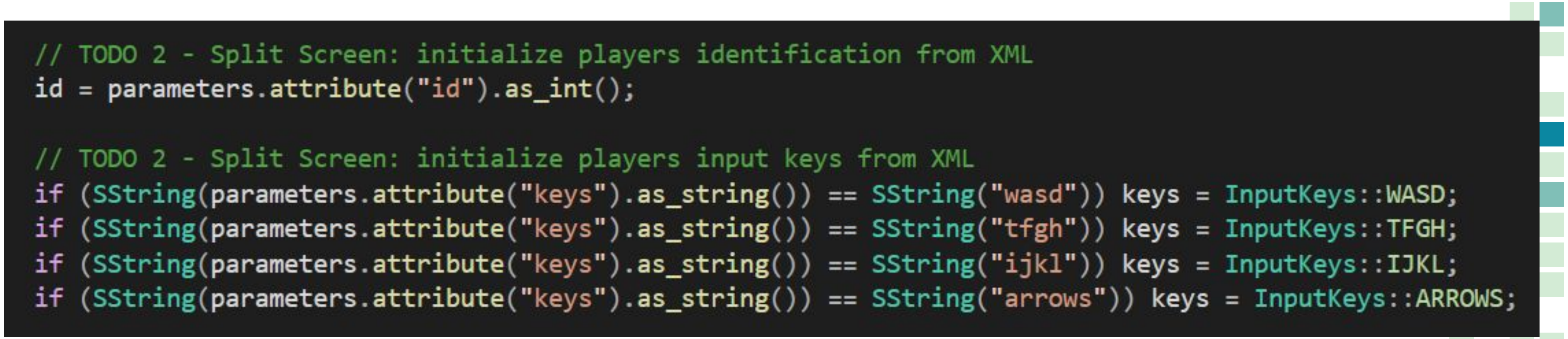
- **[Render.cpp] “Create cameras according to a viewport”**



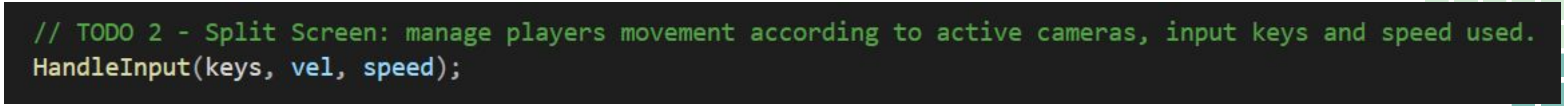
```
// TODO 1 - Split Screen: write a function to create a camera according to a given viewport and add it to the cameras list.
void Render::AddCamera(SDL_Rect viewport)
{
	Camera* camera = new Camera(viewport);

	cameras.Add(camera);
}

// TODO 1 - Split Screen: write a function to empty the cameras list.
void Render::ClearCameras()
{
	cameras.Clear();
}
```

5. HANDOUT SOLUTION – TODO 2

- **[Player.cpp] “Manage players movement and camera”**



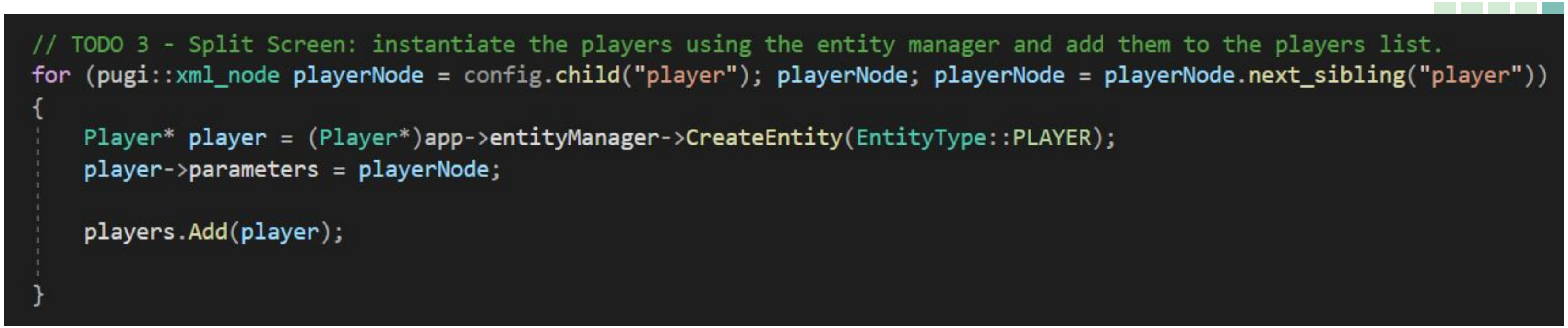
```
// TODO 2 - Split Screen: initialize players identification from XML
id = parameters.attribute("id").as_int();

// TODO 2 - Split Screen: initialize players input keys from XML
if (SString(parameters.attribute("keys").as_string()) == SString("wasd")) keys = InputKeys::WASD;
if (SString(parameters.attribute("keys").as_string()) == SString("tfgh")) keys = InputKeys::TFGH;
if (SString(parameters.attribute("keys").as_string()) == SString("ijkl")) keys = InputKeys::IJKL;
if (SString(parameters.attribute("keys").as_string()) == SString("arrows")) keys = InputKeys::ARROWS;
```



```
// TODO 2 - Split Screen: manage players movement according to active cameras, input keys and speed used.
HandleInput(keys, vel, speed);
```

5. HANDOUT SOLUTION – TODO 3

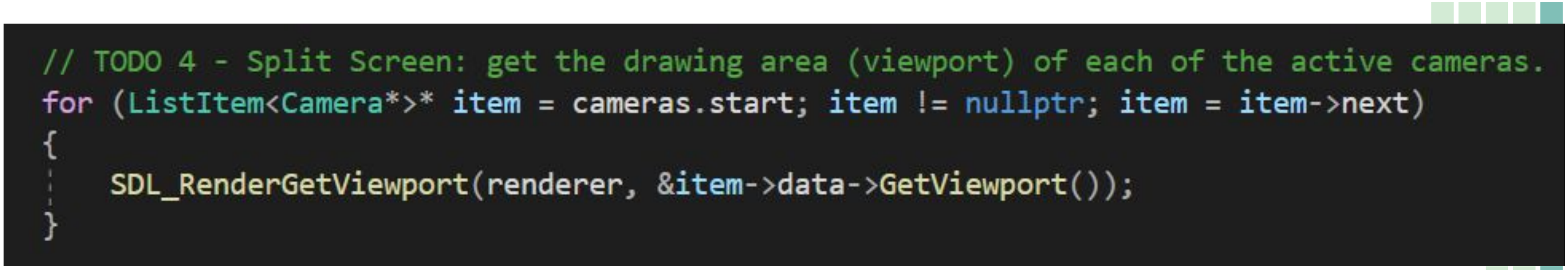
- **[Scene.cpp] “Instantiate the players in the scene”**



```
// TODO 3 - Split Screen: instantiate the players using the entity manager and add them to the players list.
for (pugi::xml_node playerNode = config.child("player"); playerNode; playerNode = playerNode.next_sibling("player"))
{
    Player* player = (Player*)app->entityManager->CreateEntity(EntityType::PLAYER);
    player->parameters = playerNode;

    players.Add(player);

}
```

5. HANDOUT SOLUTION – TODO 4

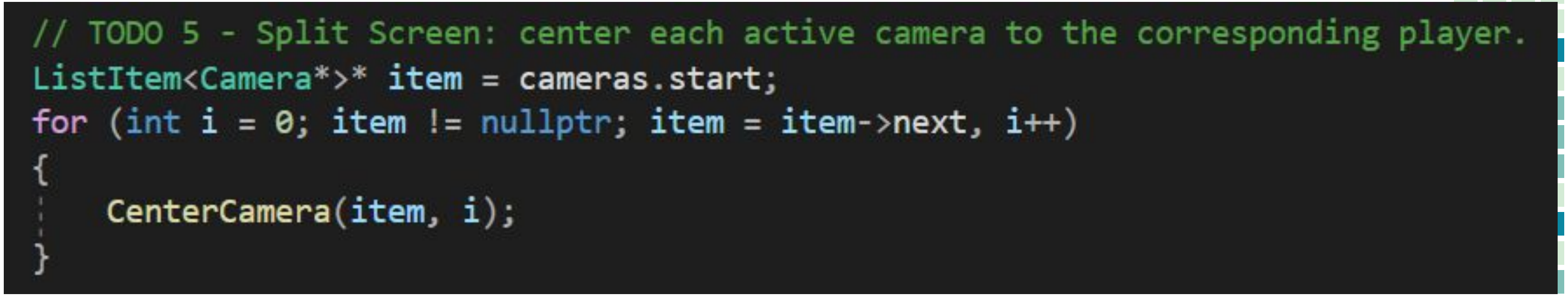
- **[Render.cpp] “Get the viewport of each active camera”**



```
// TODO 4 - Split Screen: get the drawing area (viewport) of each of the active cameras.
for (ListItem<Camera*>* item = cameras.start; item != nullptr; item = item->next)
{
    SDL_RenderGetViewport(renderer, &item->data->GetViewport());
}
```

5. HANDOUT SOLUTION – TODO 5

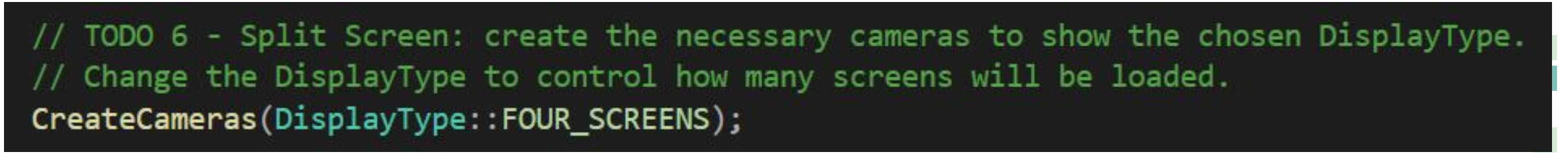
- **[Render.cpp] "Center each active camera to a player"**



```
// TODO 5 - Split Screen: center each active camera to the corresponding player.
ListItem<Camera*>* item = cameras.start;
for (int i = 0; item != nullptr; item = item->next, i++)
{
    CenterCamera(item, i);
}
```

5. HANDOUT SOLUTION – TODO 6

- **[Scene.cpp] "Create all the necessary cameras"**



```
// TODO 6 - Split Screen: create the necessary cameras to show the chosen DisplayType.
// Change the DisplayType to control how many screens will be loaded.
CreateCameras(DisplayType::FOUR_SCREENS);
```

6. POSSIBLE IMPROVEMENTS

- **Problems of this implementation in SDL2**

➢ Number of screens and their position is **hardcoded.**

➢ Four players appear regardless of how many cameras there are.

➢ There's a little visual bug on the edges of the cameras if you pay attention to it.

6. POSSIBLE IMPROVEMENTS

- **Features you could add to your Split Screen**

➢ Ability to change screen display mode mid-game.

➢ Render line, circle and text taking into account viewport margins.

➢ An algorithm that makes **n** screens for **n** players (💀)

➢ Can you think of anything else? :D

7. CONCLUSIONS

- **Advantages of the Split Screen**

  - ➢ Players can see where the other players are.
  - ➢ Only one console needed.
  - ➢ Only one copy of the game needed.
  - ➢ No need for internet access.
  - ➢ No internet connection problems.

7. CONCLUSIONS

- **Disadvantages of the Split Screen**

  - Smaller screens.
  - Performance issues.
  - More distraction, especially from game sounds.
  - Lower active resolution for each player.

7. CONCLUSIONS

- **Split Screen in our project**

  ➢ It’s hard to manage multiple screens in SDL.

  ➢ Think if it really is necessary to your game.

SDL
Simple Directmedia Layer

8. REFERENCES

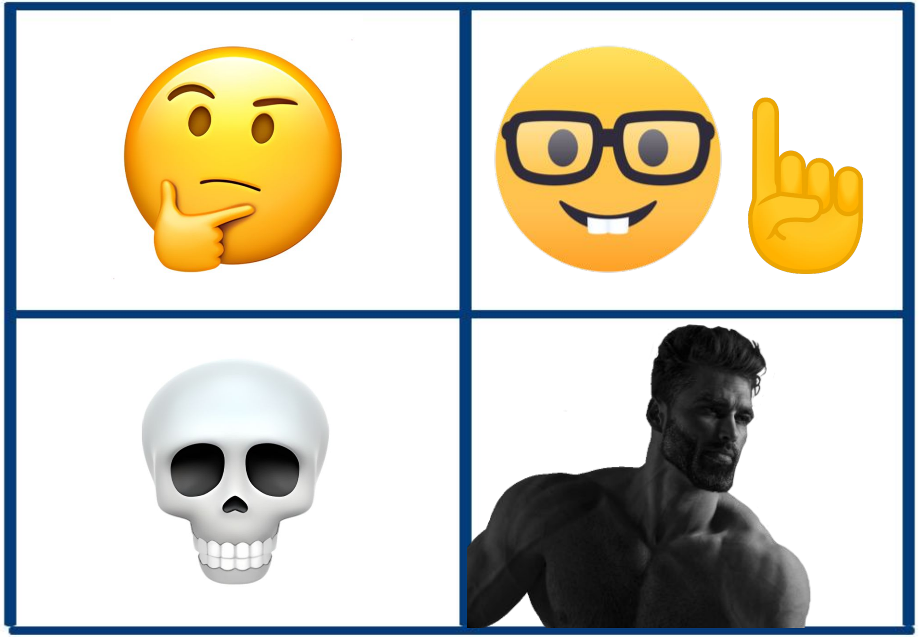
- **All the information is on my GitHub**

francesctr4

Repository

Website

ANY QUESTIONS?



THANK YOU VERY MUCH!