Sly Contest

Elisa Goloviatinski, David Jaton, Sébastien Mendes, Eddy Naddeo

Résumé

Sly Contest est un projet élaboré dans le cadre du module "RS430.100.23.2281 - Projet P2 SA" à la HE-Arc. Il se présente comme un jeu de plateforme multijoueur où l'objectif est de créer un niveau composé de plates-formes et de pièges que l'adversaire doit compléter dans un laps de temps défini. Une fois la phase de construction du niveau achevée, les deux joueurs doivent parcourir le niveau créé par l'autre, puis accomplir leur propre niveau pour prouver sa faisabilité. Il est également possible de sauvegarder les niveaux créés pour une expérience solo ultérieure. Si un niveau construit se révèle impossible, le créateur perd des points à la fin de la partie. Le joueur avec le plus de points à la fin remporte la partie. Des ajustements supplémentaires sont nécessaires, notamment l'ajout d'une plus grande variété de pièges et de plates-formes pour rendre les niveaux encore plus complexes et étendre les possibilités du jeu.

1. Introduction

Le monde du jeu vidéo est un environnement dynamique et en constante évolution. Dans ce vaste domaine, le jeu de plateforme 2D a su séduire et captiver les joueurs, non pas par sa complexité, mais par son gameplay. Sly Contest s'inscrit dans cette lignée de jeux vidéo visant à offrir une expérience de jeu dynamique et amusante, avec une partie de réflexion. Un jeu où les parties s'enchaînent sans que les joueurs voient le temps passer.

Sly Contest est un jeu multijoueur se jouant en un contre un. Dans une première phase, l'objectif est de construire un niveau pour son adversaire. Il doit être suffisamment compliqué pour que celui-ci ait du mal à le terminer. Cependant, attention, s'il n'y parvient pas, une pénalité est appliquée ! Une fois la construction terminée, il faut réussir à terminer le niveau de l'adversaire pour gagner des points. Une fois le niveau de son adversaire complété, si celui-ci n'a pas réussi, il faudra compléter son propre niveau pour prouver sa faisabilité.

Ce rapport présente les détails du développement de Sly Contest, détaillant chaque étape de sa conception. Commençant par une présentation du jeu de plateforme et de nos objectifs pour Sly Contest. Ensuite, l'analyse du projet en détaillant les inspirations pour sa conception, le cahier des charges, les spécifications détaillées et les risques possibles.

Vient ensuite la conception et tous les détails de son implémentation, en commençant par l'environnement dans lequel il a été développé. Ensuite, l'explication de la partie multijoueur et réseau en détail, puis la partie gameplay et interface utilisateur. Enfin, la présentation des limitations rencontrées et des perspectives pour Sly Contest.

2. Etat de l'art

Le jeu vidéo est un domaine vaste, riche en genres et en catégories. Nous ne pouvons évidemment pas traiter toutes ces catégories, mais nous nous concentrerons sur l'évolution d'une en particulier : le jeu de plateforme. Sly Contest étant un jeu de plateforme 2D avec une composante de level design, nous explorerons le début du jeu de plateforme, son évolution, et ensuite parlerons de notre jeu.

2.1. Le jeu de plateforme

Pour commencer, expliquons ce qu'est un jeu de plateforme. C'est l'un des genres de jeux vidéo les plus répandus. Les premiers jeux de plateformes étaient en 2D, incarnant un héros devant compléter des niveaux en avançant généralement vers la droite de l'écran, évitant des obstacles, sautant par-dessus des ennemis et se déplaçant de plateforme en plateforme. La série la plus connue de jeux de plateforme est Super Mario Bros, qui a connu un succès continu depuis sa sortie sur la NES jusqu'à ses dernières itérations sur la Switch avec Super Mario Wonder.

2.2. Début du jeu de plateforme

Le terme "jeu de plateforme" apparaît dans la presse pour la première fois en 1989. Cependant, des jeux entrant dans cette catégorie existaient bien avant l'apparition de ce terme. En 1978, le jeu "Frogs" sort dans les salles d'arcade. Développé par Sega, il met en scène une grenouille posée sur une plateforme, devant sauter pour attraper des insectes. Bien que ne correspondant pas exactement aux jeux de plateforme d'aujourd'hui, certains éléments étaient déjà présents.

Figure 2.2.1 - "Image du jeu Frogs"

En 1981 sort "Donkey Kong". Développé par Nintendo, il est considéré par beaucoup comme le premier jeu pouvant être catégorisé comme un jeu de plateforme. Les premiers éléments sont là, les obstacles à éviter, les plateformes sur lesquelles il faut avancer pour arriver au bout du niveau. Cependant il n'a rien inventé, les éléments présents dans Donkey Kong existait déjà dans d'autres jeux d'arcade, cependant il est le premier à réussir à les rassembler et les rendre cohérents pour en faire non pas un jeu de plateforme mais un

excellent jeu d'action. Il aura malgré tout permis un grand pas en avant dans le jeu de plateforme.
Le premier grand succès du genre a été "Super Mario Bros", sorti en 1985 sur Famicom au Japon puis en 1987 sur Nes en Europe, démocratisant le genre.

Figure 2.2.2 - Image en jeu de "Super Mario Bros"

2.3. Le jeu de plateforme aujourd'hui

Maintenant que nous avons établi ce qu'est un jeu de plateforme et comment le genre s'est développé, regardons comment il a évolué dans les productions actuelles. Avec l'avènement de la 3D dans le jeu vidéo, il a été nécessaire de trouver des solutions pour adapter le gameplay, car ce qui fonctionnait en 2D ne fonctionnerait pas nécessairement en 3D. Mario a une fois de plus montré la voie avec le jeu Super Mario 64, orientant son gameplay vers l'exploration et les objectifs à remplir dans des niveaux entièrement ouverts.

D'un autre côté, le jeu Crash Bandicoot a montré une autre façon d'adapter le gameplay, avec des niveaux en couloirs où il faut éviter des trous et des obstacles, le but restant d'atteindre la fin du niveau. Les jeux des années suivantes se sont inspirés de ces deux modèles. À partir de la Wii, une tendance s'est manifestée après la sortie de New Super Mario Bros Wii pour les jeux de plateforme 2D. Ces derniers suivaient toujours le même modèle de l'époque, mais ajoutaient des éléments de gameplay tels que la coopération à quatre ou plus de mécaniques de jeu. Ainsi, deux tendances distinctes ont émergé pour les jeux de plateforme. 2D : l'action et la réactivité sont au cœur du gameplay, tandis que pour les jeux de plateforme 3D, l'accent est mis sur l'exploration et l'achèvement d'objectifs pour terminer un niveau.

Figure 2.3.1 Premier niveau de “Super Mario 64”

Figure 2.3.2 Niveau de “Crash Bandicoot”

2.4. Sly Contest

Notre jeu s'inscrit dans la catégorie des jeux de plateforme. En effet, les joueurs devront compléter des niveaux en sautant de plateforme en plateforme tout en évitant des pièges et des obstacles. Pour enrichir le gameplay, nous avons décidé d'ajouter une composante d'édition de niveaux. Deux joueurs doivent d'abord construire un niveau pendant 2 minutes, puis jouer sur celui que leur adversaire a conçu pour eux. S'ils réussissent le niveau de leur adversaire, ils devront ensuite réussir leur propre niveau pour prouver sa faisabilité. En cas d'échec à terminer un niveau, une pénalité de points sera appliquée.

L'inspiration pour notre jeu provient principalement de deux autres titres. Tout d'abord, Super Mario Maker nous a influencés pour la technique d'édition de niveau basée sur une grille permettant de placer les éléments. Ensuite, Ultimate Chicken Horse a contribué à l'aspect course de complétion de niveau. La combinaison de ces deux jeux nous a donné l'idée de créer Sly Contest.

Ce projet nous a offert une précieuse expérience dans le développement d'un jeu vidéo, utilisant le moteur Godot. De plus, nous avons acquis des compétences en gestion du multijoueur en ligne et en création de système d'édition de niveaux.

3. Analyse

Sly Contest est un jeu de plateforme 2D multijoueur compétitif dynamique pour 2 joueurs. A chaque début de partie, les joueurs ont un temps imparti pour pouvoir créer un niveau réalisable avec plateformes et pièges. Une fois le temps écoulé, chaque joueur doit parcourir le niveau de l'autre et, une fois celui-ci réussi, valider le sien. Le but est donc de créer un niveau assez difficile pour que l'autre joueur ne réussisse pas à finir avant, tout en s'assurant de pouvoir réussir à le finir.

Les inspirations principales sont "Super Mario Maker" pour la partie édition et sauvegarde de niveaux et "Ultimate Chicken Horse" pour la partie duels rapides entre joueurs.

Figure 3.1 - éditeur de niveau de "Super Mario Maker"

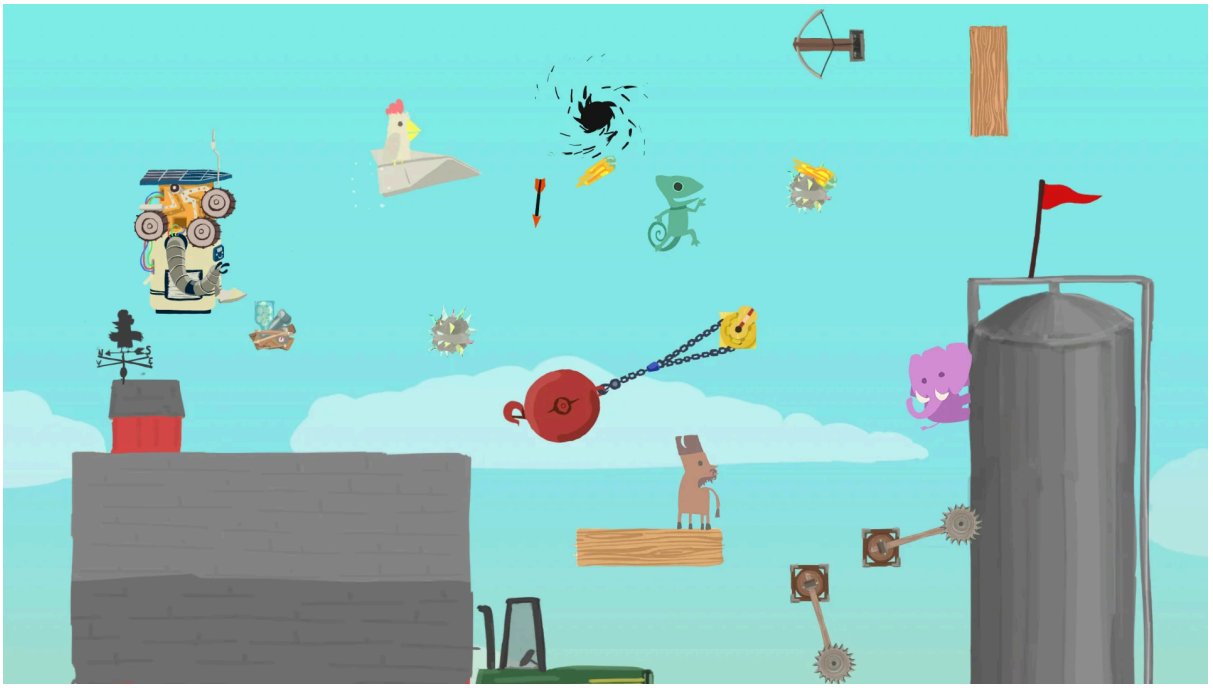


Figure 3.2 - partie standard de “Ultimate Chicken Horse”

La motivation principale derrière ce projet est d’apprendre à prendre en main un moteur de jeu et comprendre comment il fonctionne.

Objectifs:

- Principaux
  - Platformer 2D.
  - Éditeur de niveaux avec système de validation de niveaux.
  - Multijoueur versus (course).
  - Avoir de l’aléatoire.

- Secondaires
  - Plusieurs maps.
  - Système de sauvegarde de niveaux.
  - Rejouabilité solo des niveaux.
  - Écran séparé (En réseau).

Besoins:

- Généraux
  - Moteur de jeu.
  - Réseau pour le multijoueur (si pas déjà inclus par le moteur de jeu).
  - Assets pour le visuel/l’audio.

- UI
  - Différents menus (options, lobby, sélection de niveaux, score etc.).

- Options (plein écran, ajustement du son/de la musique).
- Onglet plateformes/pièges.

- Graphic/Sound design
  - Différents SFX (par exemple quand le joueur saute).
  - Indication visuelle de quel bloc/piège sélectionné.
  - Indication visuelle si le bloc/piège sélectionné a été placé ou non.
  - Différenciation visuelle du point de départ/arrivée.
  - Différenciation visuelle des deux joueurs.
  - Affichage timer, indication addition de temps.

- Mécaniques de gameplay
  - Physiques éléments niveau (collisions).
  - Physiques joueurs, confort de jeu (coyote time, buffer time, wall jump etc.).
  - Système de grille.
  - Système de placement/destruction des blocs et pièges.
  - Système de sauvegarde/chargement de niveaux.
  - Système de score (sauvegarde entre parties, pénalité etc.).

- Réseau
  - Serveur.
  - Synchronisation placement/destruction blocs/pièges.
  - Ajout/suppression des joueurs du lobby.
  - Code pour pouvoir rejoindre un lobby spécifique.

Spécifications:

- Editeur de niveau dans le style de “Super Mario Maker”
  - Basé sur une grille pour y placer divers objets (plateformes, pièges en tout genre).
  - Sauvegardes des niveaux ainsi créés.
  - Des départs et arrivées générés aléatoirement à chaque partie.

- Des courses en duels rapides dans le style de “Ultimate Chicken Horse”
  - Versus en ligne.
  - Vue sur l'écran adverse tout au long de la partie.
  - Système de validation de niveau dynamique.
    - Chaque joueur tente de terminer le niveau qu'a concocté son adversaire.
    - Le premier joueur à passer la ligne d'arrivée doit ensuite jouer son propre niveau.
    - Si aucun joueur ne passe la ligne d'arrivée pour un niveau donné le créateur reçoit une pénalité.

- Mode de jeu
  - Un mode créatif où tout est permis, nombre d'objets et sélection illimitée.

3.1. Faisabilité

Sly Contest est loin d'être le premier platformer 2D multijoueur avec éditeur de niveaux, comme il est possible de voir avec nos inspirations principales. Les contraintes principales seront donc plutôt de temps, ainsi que ceux du moteur de jeu et du langage utilisé pour celui-ci. Cependant, considérant sa nature en tant que jeu, un élément important à considérer est surtout la jouabilité et l'attirance de celui-ci si on devait hypothétiquement le publier quelque part. Ceci va surtout dépendre des mécaniques implémentées ainsi que de la façon dont elles seront implémentées et du confort du gameplay dans notre cas vu qu'on a pas d'artiste parmi nous pour pouvoir attirer un joueur potentiel par une esthétique intéressante.

3.2. Risques

- Légaux:
  - IP des assets
    - N°1.
    - Si asset protégé et pas le droit de l'utiliser peut causer des ennuis par la suite.
    - Improbable (dans le cadre d'un projet universitaire).
    - Impact modéré.
    - Criticité modérée.
    - Essayer d'utiliser assets du domaine publique ou utiliser IA.
    - Enlever et remplacer les assets affectés du jeu sinon.

- Physiques:
  - Météo
    - N°2.
    - Peut causer des retards de trains, pire des cas membres de l'équipe doivent rester chez eux.
    - Probable.
    - Impact faible.
    - Criticité faible.
    - Établir tâches en avance, communication avec les membres de l'équipe.
    - Possible que des heures additionnelles soient nécessaires pour compenser le retard.
  - Catastrophe/décès
    - N°3.
    - Impossibilité d'accéder au matériel physique, réorganisation tâches nécessaire.
    - Improbable.
    - Impact grave

- Criticité faible.

- Utiliser système de contrôle de versions en ligne, établir liste des tâches en avance ainsi que quelles personnes sont assignées à celles-ci.
- Changer les assignations des tâches dans le pire des cas si possible, heures additionnelles pour compenser retard.

- Intangibles:
  - Compétences
    - N°4.
    - Va limiter ce qui est possible à faire dans le temps imparti du projet, ainsi que les tâches que les membres de l'équipe sont capables de faire.
    - Très probable.
    - Impact grave.
    - Criticité grave.
    - Éviter de mettre objectifs/créer spécifications plus haut qu'un cran de ce que les membres de l'équipe sont capables.
    - Sinon se focaliser sur les objectifs principaux.

  - Relations/vie privée
    - N°5.
    - Impact sur communication dans l'équipe ainsi que capacité à accomplir les tâches assignées.
    - Probable.
    - Impact modéré.
    - Criticité modérée.
    - Communication importante, réassignation des tâches.
    - Possible que des heures additionnelles soient nécessaires pour compenser le retard.

- Techniques:
  - Moteur de jeu
    - N°6.
    - Dépendant du moteur du jeu et de son langage de programmation utilisé, peut être difficile de trouver de la documentation ou changer de moteur de jeu/langage de programmation si nécessaire.
    - Probable.
    - Impact modéré.
    - Criticité modérée.
    - Choisir un moteur de jeu avec assez de documentation ainsi qu'un langage répandu, bien commenter le code.

  - Serveur
    - N°7.

- Possible que serveur soit hors service.
- Probable.
- Impact très grave.
- Criticité grave.
- S'assurer qu'il soit possible d'utiliser un autre serveur, notamment un en local.

N°	Nom	Probabilité	Impact
1	IP des assets	Improbable	Modéré
2	Météo	Probable	Faible
3	Catastrophe/décès	Improbable	Grave
4	Compétences	Très probable	Grave
5	Relations/vie privée	Probable	Modéré
6	Moteur de jeu	Probable	Modéré
7	Serveur	Probable	Très grave

Figure 3.2.1 - résumé des risques

3.3. Planification

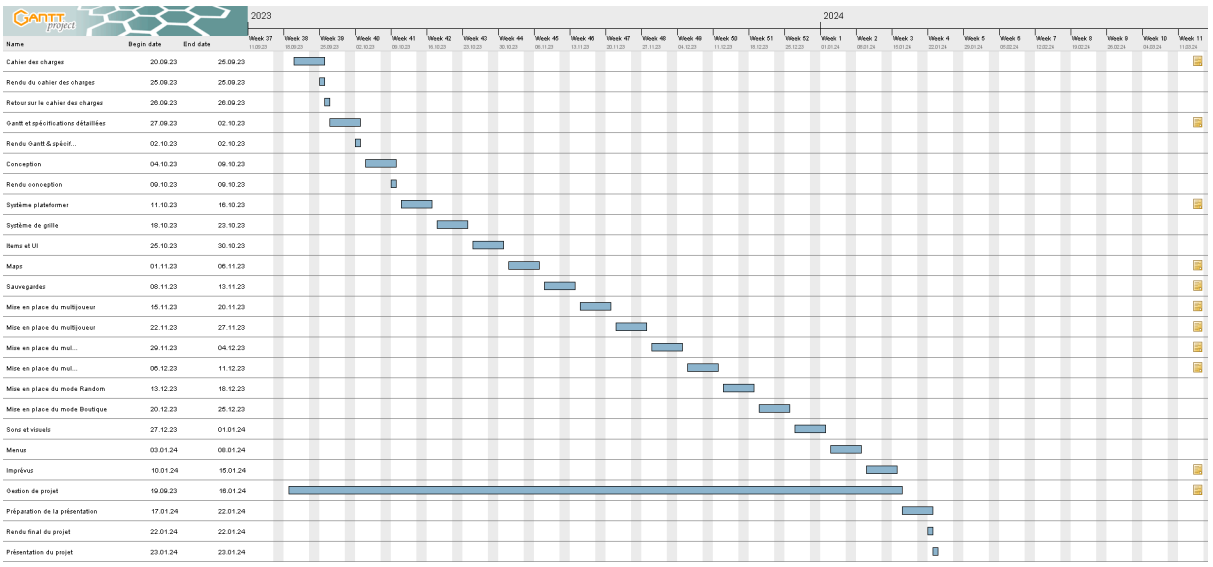


Figure 3.3.1 - diagramme de Gantt

3.4. Technologies utilisées

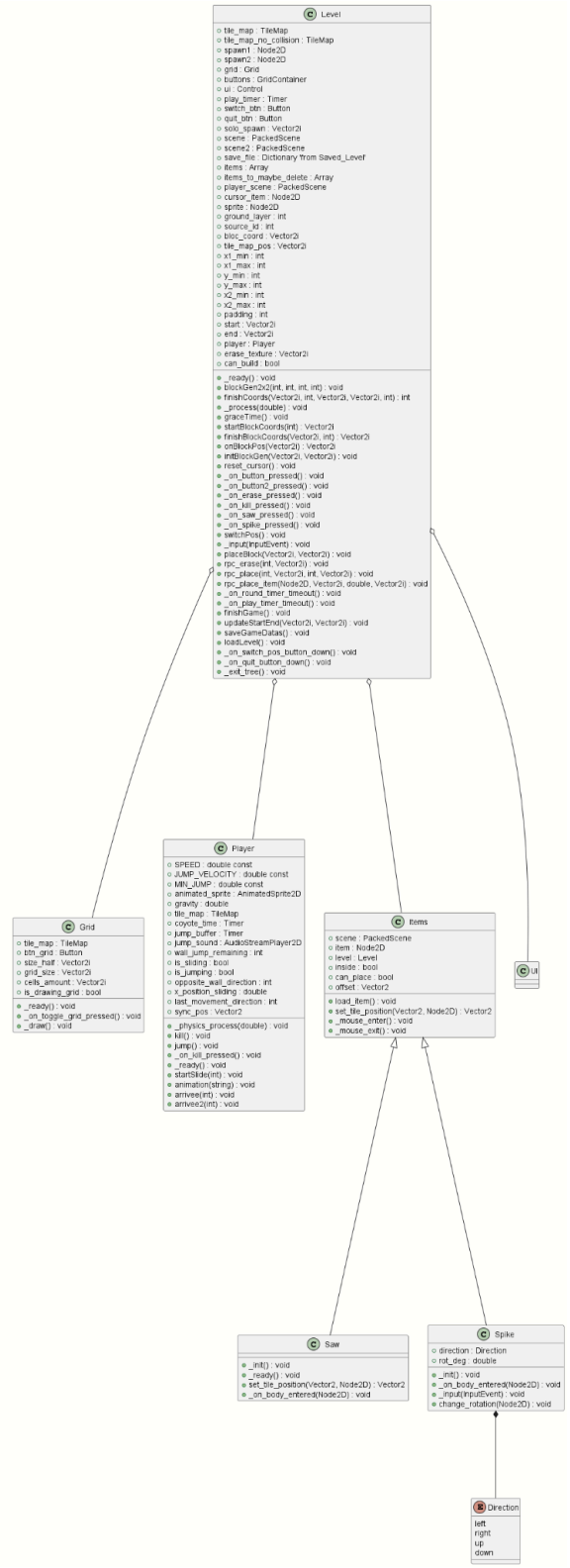
Pour le moteur de jeu, on a décidé d'utiliser Godot (4.x) qui est gratuit et open-source, bien qu'il ne soit pas aussi utilisé ou connu que Unity ou Unreal Engine. On a donc utilisé

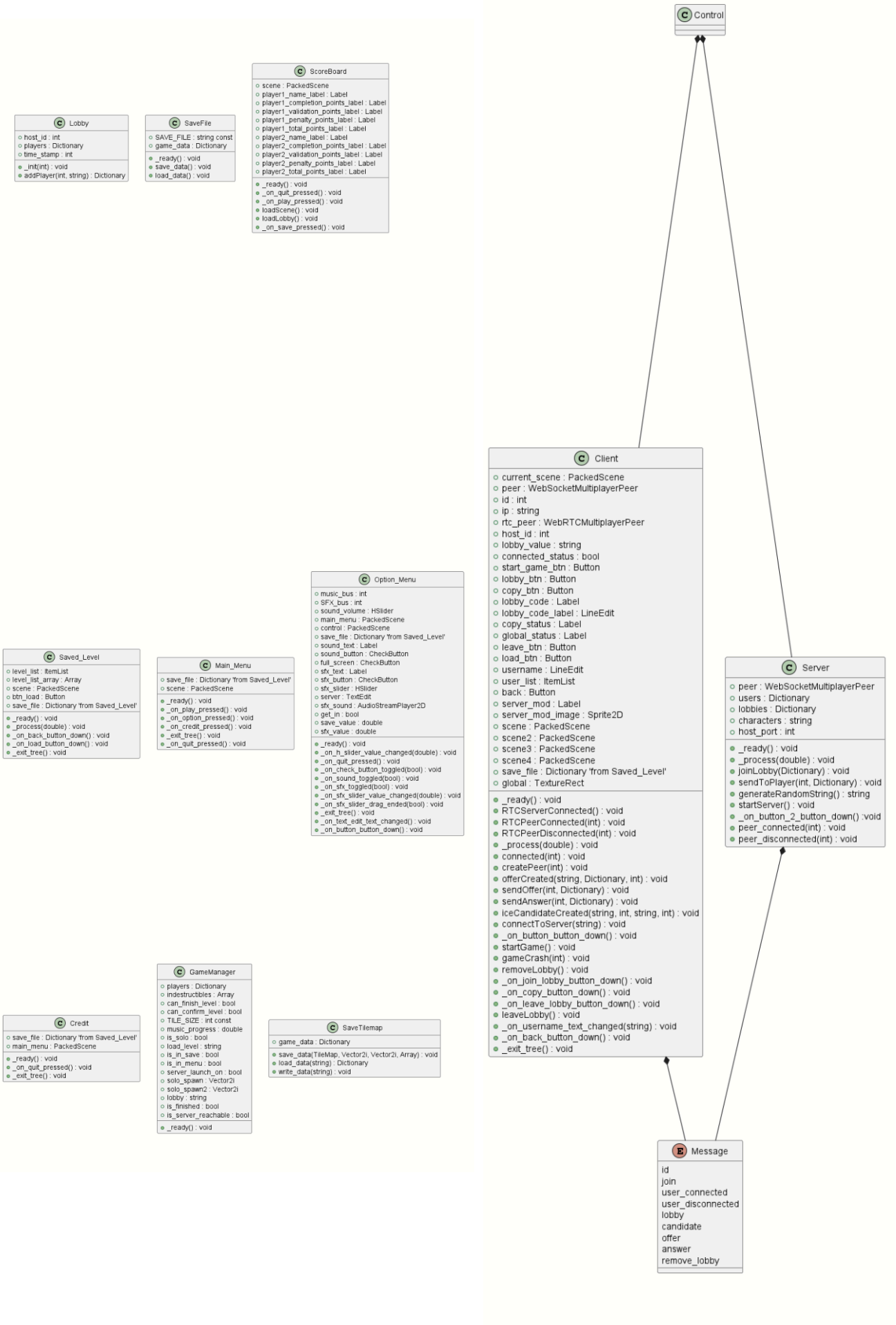
GDScript comme langage de programmation mais il aurait été aussi possible d'utiliser C++ ou C#. Sa syntaxe est similaire aux langages de type Python, servant comme une couche d'abstraction par dessus le code C++ de Godot. Il a aussi le plus de documentation pour Godot, ce qui permet une prise en main plus rapide. Pour la partie réseau, Godot implémente WebRTC.

Certains assets utilisés sont du domaine public et pour le reste on les a générés grâce à l'IA (Bing Image Creator) vu qu'il était difficile de trouver des assets d'une qualité satisfaisante qui resteraient dans le style du jeu..

4. Conception

Voici le diagramme UML du jeu. Les blocs dans notre cas ne représentent pas des classes mais des scènes. Le moteur de jeu Godot fonctionne par scène sur lesquels on place des éléments, auxquels on peut attacher des script dans lesquels on décrit les actions à réaliser en cas d'interaction avec les éléments.




Control
ScoreBoard
scene : PackedScene
player1_name_label : Label
player1_completion_points_label : Label
player1_validation_points_label : Label
player1_penalty_points_label : Label
player1_total_points_label : Label
player2_name_label : Label
player2_completion_points_label : Label
player2_validation_points_label : Label
player2_penalty_points_label : Label
player2_total_points_label : Label
_ready() : void
_on_quit_pressed() : void
_on_play_pressed() : void
loadScene() : void
loadLobby() : void
_on_save_pressed() : void
Lobby
host_id : int
players : Dictionary
time_stamp : int
_init(int) : void
addPlayer(int, string) : Dictionary
SaveFile
SAVE_FILE : string const
game_data : Dictionary
_ready() : void
save_data() : void
load_data() : void
Client
current_scene : PackedScene
peer : WebSocketMultiplayerPeer
id : int
ip : string
rtc_peer : WebRTCMultiplayerPeer
host_id : int
lobby_value : string
connected_status : bool
start_game_btn : Button
lobby_btn : Button
copy_btn : Button
lobby_code : Label
lobby_code_label : LineEdit
copy_status : Label
global_status : Label
leave_btn : Button
load_btn : Button
username : LineEdit
user_list : ItemList
back : Button
server_mod : Label
server_mod_image : Sprite2D
scene : PackedScene
scene2 : PackedScene
scene3 : PackedScene
scene4 : PackedScene
save_file : Dictionary 'from Saved_Level'
global : TextureRect
_ready() : void
RTCServerConnected() : void
RTCPeerConnected(int) : void
RTCPeerDisconnected(int) : void
_process(double) : void
connected(int) : void
createPeer(int) : void
offerCreated(string, Dictionary, int) : void
sendOffer(int, Dictionary) : void
sendAnswer(int, Dictionary) : void
iceCandidateCreated(string, int, string, int) : void
connectToServer(string) : void
_on_button_button_down() : void
startGame() : void
gameCrash(int) : void
removeLobby() : void
_on_join_lobby_button_down() : void
_on_copy_button_down() : void
_on_leave_lobby_button_down() : void
leaveLobby() : void
_on_username_text_changed(string) : void
_on_back_button_down() : void
_exit_tree() : void
Option_Menu
music_bus : int
SFX_bus : int
sound_volume : HSlider
main_menu : PackedScene
control : PackedScene
save_file : Dictionary 'from Saved_Level'
sound_text : Label
sound_button : CheckButton
full_screen : CheckButton
sfx_text : Label
sfx_button : CheckButton
sfx_slider : HSlider
server : TextEdit
sfx_sound : AudioStreamPlayer2D
get_in : bool
save_value : double
sfx_value : double
_ready() : void
_on_h_slider_value_changed(double) : void
_on_quit_pressed() : void
_on_check_button_toggled(bool) : void
_on_sound_toggled(bool) : void
_on_sfx_toggled(bool) : void
_on_sfx_slider_value_changed(double) : void
_on_sfx_slider_drag_ended(bool) : void
_exit_tree() : void
_on_text_edit_text_changed() : void
_on_button_button_down() : void
Saved_Level
level_list : ItemList
level_list_array : Array
scene : PackedScene
btn_load : Button
save_file : Dictionary 'from Saved_Level'
_ready() : void
_process(double) : void
_on_back_button_down() : void
_on_load_button_down() : void
_exit_tree() : void
Main_Menu
save_file : Dictionary 'from Saved_Level'
scene : PackedScene
_ready() : void
_on_play_pressed() : void
_on_option_pressed() : void
_on_credit_pressed() : void
_exit_tree() : void
_on_quit_pressed() : void
Server
peer : WebSocketMultiplayerPeer
users : Dictionary
lobbies : Dictionary
characters : string
host_port : int
_ready() : void
_process(double) : void
joinLobby(Dictionary) : void
sendToPlayer(int, Dictionary) : void
generateRandomString() : string
startServer() : void
_on_button_2_button_down() :void
peer_connected(int) : void
peer_disconnected(int) : void
GameManager
players : Dictionary
indestructibles : Array
can_finish_level : bool
can_confirm_level : bool
TILE_SIZE : int const
music_progress : double
is_solo : bool
load_level : string
is_in_save : bool
is_in_menu : bool
server_launch_on : bool
solo_spawn : Vector2i
solo_spawn2 : Vector2i
lobby : string
is_finished : bool
is_server_reachable : bool
_ready() : void
Credit
save_file : Dictionary 'from Saved_Level'
main_menu : PackedScene
_ready() : void
_on_quit_pressed() : void
_exit_tree() : void
SaveTilemap
game_data : Dictionary
save_data(TileMap, Vector2i, Vector2i, Array) : void
load_data(string) : Dictionary
write_data(string) : void
Message
id
join
user_connected
user_disconnected
lobby
candidate
offer
answer
remove_lobby

5. Implémentation

L'implémentation du code sera expliquée ci-dessous. Tout d'abord, le moteur de jeu, puis la partie réseau et multijoueur, ensuite la partie gameplay, et enfin l'interface utilisateur.

5.1. Moteur de jeu

Le moteur de jeu Godot a été choisi pour l'implémentation du jeu. Il fonctionne avec des scènes. Il existe trois types de scène. Un pour l'interface utilisateur qui sera là pour implémenter le front end du jeu. Les scènes 2D sont utilisées pour les éléments du jeu en deux dimensions. Le troisième type est la scène 3D pour implémenter des éléments en trois dimensions. Les éléments placés sur ces scènes sont appelés des nœuds. Ces nœuds peuvent être imbriqués, créant ainsi un nœud parent et ses enfants qui héritent de certaines de ses caractéristiques. Cette structure est particulièrement utile pour les éléments de l'interface utilisateur, facilitant l'organisation des nœuds sur la scène.

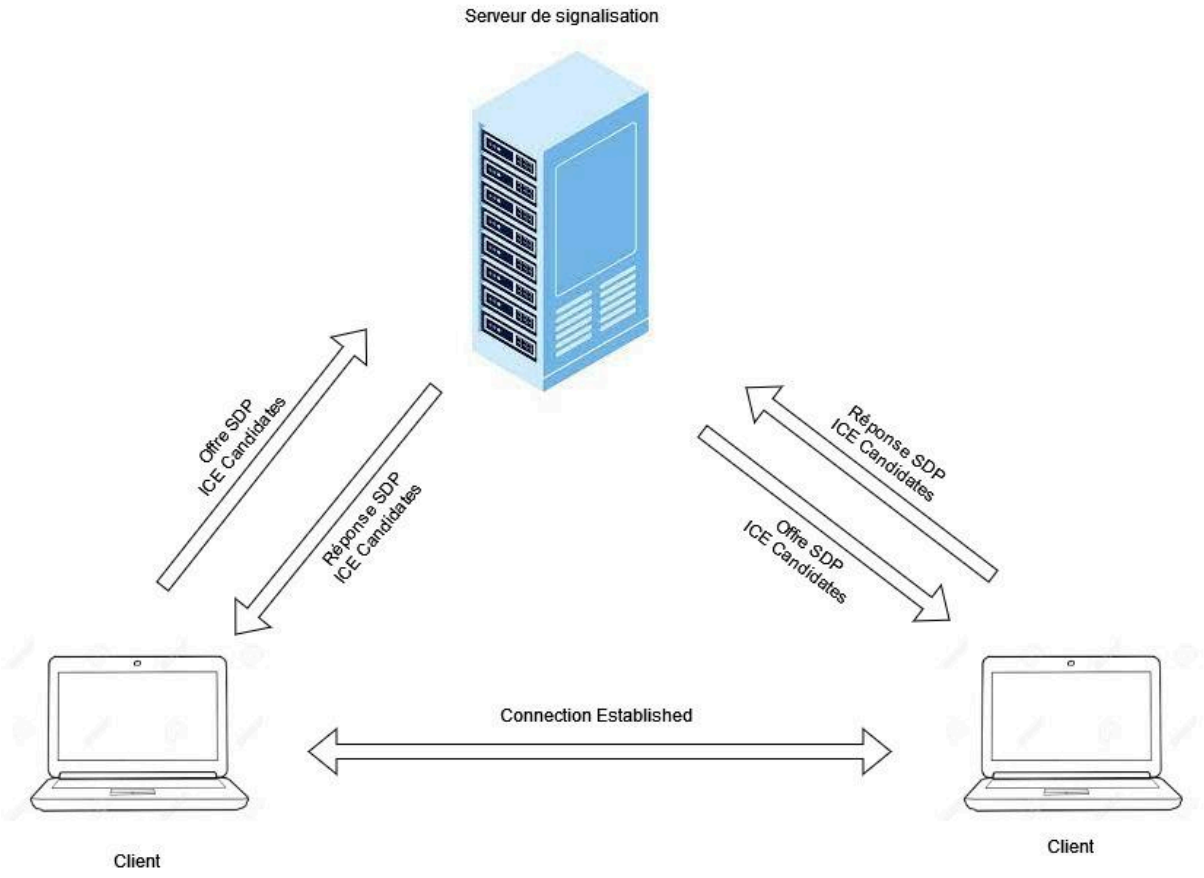
Ces nœuds ou scènes peuvent ensuite être dotés de scripts décrivant leur comportement en cas d'événements spécifiques. Chaque nœud possède des signaux auxquels il peut être connecté. On peut détailler le comportement souhaité dans la fonction associée au signal (par exemple, on_button_pressed() dans le cas d'un bouton). Les scripts Godot sont écrits dans le langage propre au moteur, le GDScript. Bien qu'il soit similaire au Python, le GDScript est optimisé pour une implémentation basée sur une architecture avec des scènes.

5.2. Network

Un script a été réalisé pour la partie serveur, organisant la mise en connexion des paires, notamment pour la gestion du lobby. Lorsqu'un joueur dans le menu principal appuie sur "Play", il se connecte au serveur et peut choisir de créer un lobby. Cela génère une requête sur le serveur pour rejoindre un lobby de deux personnes. Si aucun code spécifique n'est précisé, un lobby est créé. Lors de la création du lobby, un code est généré pour rejoindre un lobby existant. Une fois le lobby rempli, la partie peut être lancée, supprimant ainsi le lobby pour libérer de l'espace sur le serveur et éviter un débordement.

La partie réseau a été développée avec WebRTC, une technologie permettant à deux clients de se connecter en pair à pair. Bien que le principe du pair à pair n'exige pas de serveur, un serveur est nécessaire pour mettre en relation ces deux clients. Le premier client envoie une requête au serveur appelée “Offre", contenant un SDP (Session Description Protocol) qui représente la configuration multimédia du premier client, notamment s'il souhaite envoyer de l'audio, de la vidéo, ou les deux. Ensuite, le serveur envoie cette offre à l'autre client. Ce dernier répond avec une "Réponse" contenant également un SDP détaillant sa configuration multimédia. Cette réponse est envoyée au client ayant créé l'offre. En parallèle, avec leurs SDP, les clients envoient également un ICE (Interactive Connectivity Establishment)

contenant la configuration réseau. Une fois que les SDP et ICE des deux clients ont été partagés, ils peuvent se connecter en pair à pair et n'ont plus besoin du serveur.



5.3. Gameplay

Pour la partie gameplay, un nœud "player" a été créé avec son script, détaillant la physique du personnage et les touches permettant ses déplacements. Bien que les déplacements de base soient déjà implémentés lorsqu'un script est créé pour le personnage, des modifications ont été apportées, telles que la possibilité de sauter un peu en retard en quittant une plateforme, le wall jump et l'ajout d'un son au moment du saut.

Une scène "level" a également été implémentée pour la construction du niveau. Nous avons utilisé une TileMap pour les blocs de construction, permettant de délimiter des zones sur une image au format PNG et de les convertir en blocs avec des hitboxes spécifiées. Les pièges ont été implémentés sous forme de nœuds Area 2D avec une scène et un script permettant de tuer le joueur lorsqu'il les touche. Les informations nécessitant une communication entre les différentes scènes ou entre les clients sont stockées dans la scène GameManager.

5.4. Interface utilisateur

En ce qui concerne l'interface utilisateur, plusieurs scènes ont été créées. Tout d'abord, une scène "UI" a été mise en place pour la sélection des blocs et pièges pendant la phase de construction. Cette scène contient des boutons pour les éléments sélectionnables, tous organisés dans différents "Containers" pour un alignement automatique lors de l'ajout ou de la suppression d'un bouton. Cette scène est purement visuelle et ne comporte pas de script.

Les différentes scènes de menu sont composées de la même manière, tout comme les crédits et le tableau des scores.

Le menu principal possède un script permettant de changer de scène en fonction du bouton sélectionné (Options, Crédits, Play, Quit), ainsi qu'un nœud pour la gestion de la musique.

Le menu Options dispose d'un script permettant de régler le volume de la musique et des effets sonores à l'aide d'un curseur, ainsi que d'un bouton basculant. Un autre bouton basculant permet de passer en mode plein écran, et dans ce menu, il est également possible de changer le serveur auquel on tente de se connecter.

Le menu Crédits est purement visuel et affiche les informations de développement du jeu.

Le menu Scoreboard possède un script permettant de relancer une partie avec une sauvegarde des points, d'afficher les scores à la fin d'une partie et de sauvegarder les cartes pour y rejouer en solo.

.

6. Résultats

On a pu remplir tous nos objectifs, qu'ils soient principaux ou secondaires, bien que certains de ceux-ci peuvent encore être un peu plus étoffés, dont notamment la rejouabilité solo, l'aléatoire et la partie d'avoir plusieurs maps. On a aussi pu implémenter les spécifications.

Le jeu a des fonctionnalités additionnelles en dehors des spécifications dont notamment la possibilité de choisir le serveur et en démarrer un en local dans les options (auquel cas pour jouer il faut tout simplement exécuter une autre instance du jeu).

La latence ne se remarque presque pas, même lorsqu'on joue sur des réseaux différents.

La boucle de jeu marche bien et les contrôles et physiques du personnage sont confortables.

Détails de l'implémentation des spécifications:

- Editeur de niveau dans le style de "Super Mario Maker"
  - Système de grille implémenté, quelques plateformes et pièges implémentés, aurait été judicieux d'ajouter quelques autres pièges. Parfois les piques bug un peu (possibilité d'avoir plusieurs rotations d'une pique dans la même case).
  - Système de sauvegarde et chargement des niveaux fonctionnel. Les niveaux sont sauvegardés dans des fichiers locaux binaires avec extension .SLAY.
  - Départs et arrivées générés aléatoirement à chaque partie.

- Des courses en duels rapides dans le style de "Ultimate Chicken Horse"
  - Versus en ligne. Si serveur hors service, possibilité d'héberger un serveur en local. Synchronisation de l'ajout et de la suppression des plateformes et pièges ainsi que des joueurs. Latence peu visible, même quand les deux joueurs jouent sur des réseaux différents.
  - Vue sur l'écran adverse tout au long de la partie. Il aurait peut-être fallu ajouter d'autres signes distinctifs pour différencier les deux joueurs et toute suite indiquer au joueur quel écran est le sien.
  - Système de validation de niveau dynamique implémenté. Ajout d'un grace time qui reset le timer de 60 secondes si un joueur a terminé le niveau de l'autre alors qu'il y avait moins de 60 secondes qui restaient pour donner une opportunité de valider son niveau ainsi qu'à l'autre joueur de le terminer vu qu'il est possible que le niveau soit difficile.

- Mode de jeu
  - Un mode créatif où tout est permis, nombre d'objets et sélection illimitée. Il aurait été intéressant d'ajouter d'autres modes de jeu où il y aurait des restrictions sur le nombre et type de plateformes et pièges qu'on peut placer, soit avec un mode aléatoire, soit avec un mode boutique où les joueurs peuvent accumuler de l'argent pour acheter les objets qu'ils veulent.

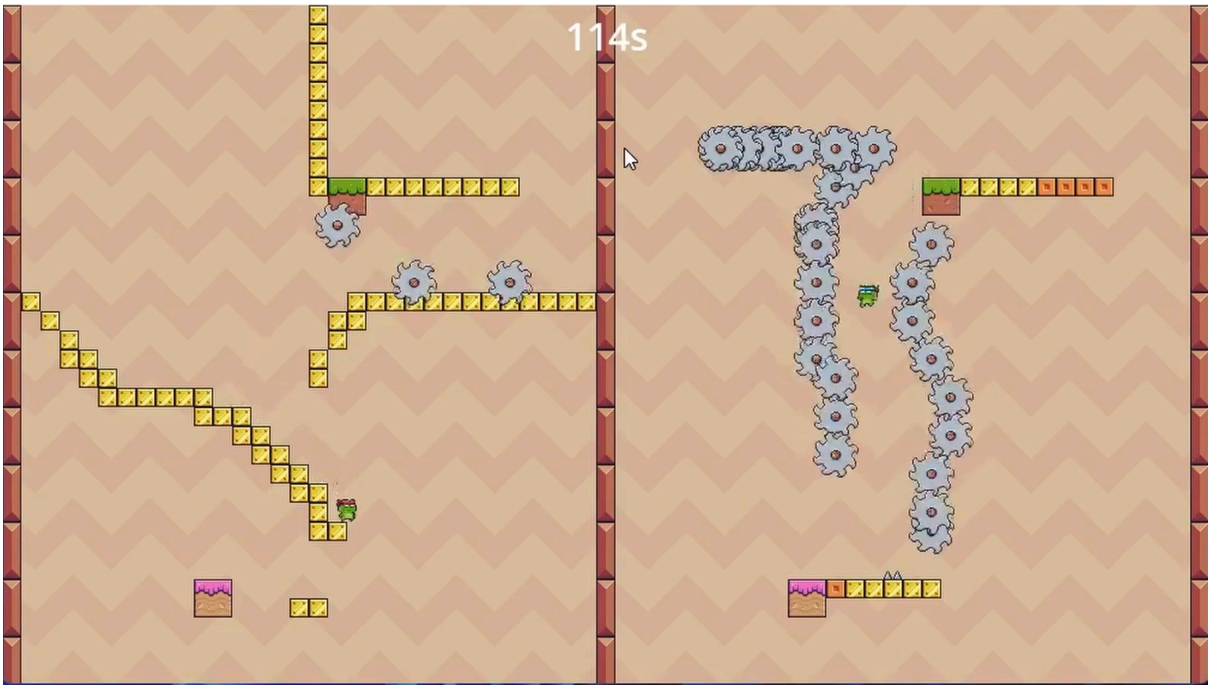


Figure 6.1 - partie standard de Sly Contest

7. Limitations et perspectives

Ce projet a surtout été limité niveau temps, ainsi que par le fait qu'on ne pouvait pas se focaliser seulement sur celui-ci. Sans ce type de limites, il aurait été possible d'ajouter plus de fonctionnalités dont notamment:

- Gameplay:
  - Plusieurs modes de jeux (mode boutique et mode aléatoire par exemple).
  - Possibilité de modifier les niveaux sauvegardés en mode solo.
  - Ajout d'autres pièges et blocs pour rendre la boucle de jeu plus intéressante et potentiellement plus complexe.
  - Ajout d'autres éléments aléatoires (plateformes préexistantes par exemple).

- Graphic/Sound design:
  - Plus de SFX.
  - Plus de différenciation visuelle entre les deux joueurs.

- Ajustements:
  - Résolution de bugs comme celui des piques.
  - Plus de tests pour trouver le maximum de bugs que possible.

Il est aussi intéressant de comparer ce jeu à ses inspirations et voir en quoi il diffère:

- Comparaison avec "Super Mario Maker":
  - Duels entre joueurs contrairement à celui-ci.
  - Bien qu'il soit possible de partager les niveaux sauvegardés, il n'y a pas de compétition en dehors du mode multijoueur.
  - Niveaux généralement plus rapides et donc dynamiques vu qu'ils sont créés pendant un temps limité et avec un nombre limités de plateformes et pièges. Donc il n'est pas vraiment possible de les rendre aussi difficiles qu'un niveau kaizo (un type de niveau difficile remplis de pièges ou avec un placement de blocs "injuste", possible de compléter seulement par tâtonnement et maîtrise des contrôles et items) sans compter aussi pouvoir le compléter dans le temps imparti.

- Comparaison avec "Ultimate Chicken Horse":
  - Deux joueurs seulement.
  - Chacun des joueurs crée un niveau différent et ne travaille donc pas sur le même niveau.
  - Chaque partie a un niveau différent, le joueur ne revient pas sur le même niveau.

8. Conclusion

Ce projet a donné une bonne opportunité de prendre en main un moteur de jeu, notamment Godot qui est moins connu que Unity ou Unreal Engine mais est un projet open-source. La partie gestion multijoueur en ligne a aussi ajouté une dimension additionnelle au projet vu que ce qui marchait en local ne marchait pas forcément en multijoueur et il fallait donc faire attention à ce que les objets et joueurs soient bien synchronisés.

On a pu respecter le cahier des charges et remplir tous nos objectifs, que ce soit les objectifs principaux ou secondaires.

Le jeu à l'heure actuelle, bien qu'il n'a pas autant de fonctionnalités qu'il aurait pu avoir, a déjà une boucle de jeu attirante et il est donc facile de jouer plusieurs parties successives. Les contrôles du personnage sont aussi agréables ce qui aide au confort du jeu.

Le système de sauvegarde et chargement de niveau permet aussi d'essayer de terminer un niveau qui a été trop difficile de finir pendant la partie.

L'ajout de la possibilité d'héberger un serveur localement ajoute aussi de la longévité au jeu.

Le projet est donc un succès et Sly Contest permet d'ores et déjà d'offrir une expérience satisfaisante tout en ayant le potentiel d'être plus encore avec juste un peu plus de temps et de peaufinement.

Table des figures

Bibliographie

« GDScript Reference ». *Godot Engine Documentation*, https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/gdscript_basics.html. Consulté le 19 janvier 2024.

Audureau, W. (2012, 01 04). *Aux sources du jeu de plateforme*. Récupéré sur merlanfrit.net: https://www.merlanfrit.net/Aux-sources-du-jeu-de-plateforme Consulté le 20 janvier 2024.

Campus des écoles. (2019, 05, -). *Les grands genres du jeu vidéo : le platformer*. Récupéré sur Campus des écoles: https://www.campus-des-ecoles.fr/design-digital/secteur-jeu-video/genres/platformer/ Consulté le 20 janvier 2024.

heyletscode. (2020, 09 05). *How Does WebRTC Work? Seriously, How?* Récupéré sur Youtube: https://www.youtube.com/watch?v=SsN4gl_wV_8 Consulté le 20 janvier 2024.