

Conception

Type :	conception
Nom du projet :	A déterminer
Auteur :	Pierre Goacolou
Version :	1.0

Je doit utiliser les types abstraits de données

Je dois respecter le cahier des charges

Fonctionnement

Dans ce jeu, seules les flèches et la touche `[espace]` sont utilisés.

Le but du jeu est d'amener la balle au drapeau de fin de niveau. Pour y arriver vous pouvez choisir dans la première phase, l'orientation et la force initiale à l'aide des flèches. Vous pouvez ensuite appuyer sur espace pour lancer la balle. Dans la seconde phase où la balle est lancée, vous devez influencer si nécessaire la vitesse de la balle avec un coup de boost (touche espace). Le boost ne change pas la direction mais la vitesse de la balle, a vous de choisir le bon moment (position, direction de la balle) pour donner les boosts. Tapotez simplement la touche espace, ne laissez la touche enfoncée.

La simulation physique se base uniquement sur les collisions de type Balle-Ligne.

Multijoueur

Pour jouer en multijoueur vous devez être sur le même réseau LAN. L'hôte de la partie doit commencer une partie normale. Les clients doivent aller dans le menu multijoueur et entrer dans la case de l'IP l'adresse IP LAN de l'hôte et ensuite se connecter. Ne changez pas le port.

Pour connaître votre ip LAN faites dans le terminal la commande `ip a`

```
[pierre@yolo jeu]$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
   link/ether d0:39:57:27:86:bf brd ff:ff:ff:ff:ff:ff
   inet 192.168.10.107/24 brd 192.168.10.255 scope global noprefixroute wlan0
       valid_lft forever preferred_lft forever
   inet6 2a01:e0a:fc3:e980:7874:9dbf:394d:1c9c/64 scope global dynamic noprefixroute
       valid_lft 85979sec preferred_lft 85979sec
   inet6 fe80::1479:d6c4:36e8:5d8f/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Niveaux

Les niveaux sont au format SVG inkscape.

- Le layer "Main" contient les lignes solides et les objets.
 - l'objet contenant l'id "start" indique le point d'apparition
 - les objets path indique les lignes. pas d'arrondis
 - La position de l'objet avec l'id `start` sert de point d'apparition
- Le layer "Bg" contient les lignes dessinés en arrière-plan.
- Le layer "Kill" contient les lignes qui au contact tue les joueurs
- Le layer "Goal" contient les lignes qui au contact fait passer le joueur au niveau suivant En jeu, les niveaux sont "parsés" dans un format optimisé pour le moteur du jeu dans le type `Level`.

Menus

Une type abstrai de donnée est créée pour gérer efficacement les menus. C'est un tableau de tableau qui représente des `MenuItems`. Ça peut représenter un texte non sélectionnable, une entrée de valeur ou un bouton. Il y a une méthode pour récupérer les events tel qu'un appui de bouton.

Affichage

L'affichage est géré par un Framebuffer et une liste d'instruction de dessin. Il y en a un pour chaque `Device`. Le type `Device` permet de faire abstraction des différents types d'affichages/terminal

- Terminal Local: `stdout` / `stdin`
- Connection Réseau: `send()` / `recv()`

Avec linux, les sockets et les flux (stdin/stdout) sont des fichiers. La structure `Device` ne contient alors que le `fd` de lecture et le `fd` d'écriture.

Types de données

```
Game = struct {
    current_party: Party
    local_device: Device

    run()
}

Party = struct {
    players: list[dict]
    game_mode: "story" | "race"
    current_level: level
}

Level = struct {
    solid_layer = list[tuple[float, float, float, float]]
    bg_layer = list[tuple[float, float, float, float]]
    simulated_entity = list[Entity]

    show(rect, RenderContext)
    update(dt)
```

```
}

Entity = struct {
    position : Vector2
    velocity : Vector2
    size : float
    rotation : float

    update(dt: float)
    show(RenderContext)
}

Vector2 = struct {
    x: float
    y: float

    add()
    sub()
    mul()
    div()
}

Framebuffer = struct {
    width: int
    height: int
    buffer : list[str]

    draw_line()
}

RenderContext = struct {
    draw_list: list

    clear()
    line()
}

MenuItem = list[type: str, text: str, ...]

Menu = struct {
    items = list[MenuItem]
    cursor : int
    x : int
    y : int
    events : list

    show(RenderContext)
    manage_input(InputManager)
    pull_event() -> int
}

Device = struct {
    recv() -> str
    send(str)
```

```
}

InputManager = struct {
    pull_event(Device) -> Key
    event_left() -> bool
}
```

Arbre Principal

```
Game.run()
+---Party.init()
    +-- Level.Parse()
```

Arbre Boucle principale

```
Game.run()
+-- Level.Update()
    +-- Entity.Update()
+-- Level.Draw()
    +-- Entity.Draw()
+-- RenderContext.Draw()
+-- Device.send()
```

Fonctions

Game::create()

Initialise le jeu

Game::run()

Lance le jeu et la boucle principale

Game::main_menu_callback()

Gère les évènements du menu principal

Game::exit_menu_callback()

Gère les évènements du menu pour quitter

Level::show()

Donne les instructions de dessin pour le niveau et ses objets

Level::show()

Donne les instructions de dessin pour le niveau et ses objets

Level::parse_path()

Grosse fonction qui interprète des commandes de dessin svg pour obtenir les lignes dont le jeu en rafolle

RenderContext::draw()

Dessine les instructions de dessin dans un framebuffer et l'affiche

InputManager::pull_event()

Retourne une touche pressée. si il y en a plus, retourne **None**

Framebuffer::draw_line()

Dessine un trait (wow)

client()

Une sorte de client telnet simplifié. (dumb terminal via le réseau)

Entity::update_player()

Fonction qui gère les actions du joueur. S'exécute que si le type de l'entité est mise sur **Player**

Physics.py/step()

Exécute un pas de simulation dans le monde.

``