Projet Multijoueur en C++ : The New Pong

Auteurs: Dounia Bakalem, Yanis Sadoun, Vasileios Filippos Skarleas

L'objectif [Yanis] - Diapo 1 (logo)

Puisqu'il n'y a rien de plus amusant pour découvrir un langage que de créer son propre jeu, nous vous présentons **The New Pong**, un jeu multijoueur développé dans le cadre du module de programmation en langage objet pour la spécialité Robotique à Polytech Sorbonne.

Nous avons donc opté pour un grand classique: Pong. Préparez-vous à renvoyer la balle, tout en perfectionnant vos compétences en C++ !

Le jeu [Yanis] - Diapo 2 (Un jue, lequel)

Afin de revisiter l'expérience Pong, l'un des tout premiers jeux vidéo d'arcade et pionnier des jeux de sport, nous avons décidé d'en développer notre propre version. Au-delà d'un simple hommage, nous y avons ajouté de nouvelles fonctionnalités pour rendre ce Pong encore plus captivant que l'original. Pour cela, quatre modes de jeu distincts ont été introduits :

1. **AI mode**
2. **Classic**
3. **Storytime mode**
4. **Fun mode**

Toutes les instructions relatives à ces modes et leurs spécificités sont détaillées dans la section: **Les différents modes**. Bonne lecture et bon amusement !

Les différents modes - Diapo 3 ou +++

Classic [Yanis]

Le concept originel de Pong s'apparente à un simulateur de ping-pong minimaliste : une balle se déplace de part et d'autre de l'écran en rebondissant sur les bords supérieur et inférieur. Chaque joueur contrôle une raquette coulissant verticalement le long du bord de l'écran. La balle rebondit différemment selon la partie de la raquette qu'elle touche. Voici les fonctionnalités incluses :

- High Score
- Game Save

Dans notre version, il n'y a pas de score maximum prédéfini ; les joueurs peuvent simplement s'entendre oralement sur un objectif à atteindre. Lorsqu'ils souhaitent arrêter, il suffit de choisir « End the game ». Ici, la motivation ultime est : ***qui fera exploser le compteur du high score et revendiquera le titre de meilleur pongiste ?***

AI mode [Yanis]

Ce mode reprend les règles du **Classic** , à la différence qu'il ne peut être joué que par un seul joueur : la raquette adverse est contrôlée par l'ordinateur. **Préparez-vous à affronter une IA tenace. Arriverez-vous à la battre, ou rejoindrez-vous la longue liste de ses victimes ?**

Storytime mode [Dounia]

Dans ce mode, deux joueurs s'affrontent sur **3 tours.** Le vainqueur est celui qui remporte le plus de tours . Chaque tour se compose de **8 points**, et c'est le premier joueur à atteindre 8 points qui gagne le tour.

Une nouveauté pimentera votre partie : des lettres tombent depuis le haut de l'écran. En les touchant, vous obtenez un point supplémentaire et vous contribuez à former un mot caché, révélant peu à peu une phrase secrète.

Fun mode [Dounia]

Ce mode s'inspire des règles du **Storytime Mode** , avec un format de 3 parties où l'objectif est d'atteindre 5 points pour remporter chaque partie. Toutefois, nous y avons glissé plusieurs surprises et easter eggs destinés à dynamiser la compétition.

Puisque nous sommes de futurs roboticiens, nous ne pouvions pas résister à ajouter une petite touche de robotique : vous verrez ainsi de mystérieux robots apparaître au cours de la partie. En les touchant, vous déclencherez des effets inédits :

- Raquette géante : votre raquette gagne temporairement en taille.
- Balle invisible : la balle disparaît momentanément, rendant la partie plus chaotique.
- Contrôles inversés : les touches de votre adversaire se retrouvent soudainement inversées.

Saurez-vous exploiter ces bonus (et pièges) pour devenir le champion incontesté du Fun Mode ?

Fonctionalites (Les petits plus) - Diapo 4

High Score [Vasilis]

Cette fonctionnalité est disponible uniquement en mode Classic . Le jeu vérifie en permanence si un joueur atteint un score supérieur au record actuel. Lorsque c'est le cas, le record est immédiatement mis à jour.

La sauvegarde est effectuée dans un fichier nommé `game_pong-highscore_849216.txt`, dont le contenu est chiffré afin de garantir l'intégrité des données et d'empêcher toute modification non autorisée. Ce fichier contient uniquement le dernier high score ainsi que le nom du joueur correspondant.

Voici l'algorithme qui détermine si quelqu'un a fait un nouveau highscore:

```
// High score logic
if (player1->get_user_score() >= last_highscore || player2-
>get_user_score() >= last_highscore)
        {
            last_highscore = (player1->get_user_score() >= player2-
>get_user_score()) ? player1->get_user_score() : player2-
>get_user_score();
```

```
            strncpy(last_highscore_name, ((player1->get_user_score() >=
player2->get_user_score()) ? player1->get_user_name() : player2-
>get_user_name()).c_str(), 19);
            last_highscore_name[19] = '\0';
        }
```

Game Save [Vasilis]

Envie de faire une pause et de retenter de battre le record un peu plus tard ? Avec la fonctionnalité de Game Save , vous pouvez sauvegarder l'état de votre partie et la reprendre quand vous le souhaitez. Là encore, le chiffrement est appliqué pour garantir l'intégrité des données.

Game save logic

```
SaveState saveState;
saveState.score1 = player1->get_user_score();
saveState.score2 = player2->get_user_score();
saveState.paddle1_y = racket1->get_pos_y();
saveState.paddle2_y = racket2->get_pos_y();
saveState.ball_x = mBall->get_pos_x();
saveState.ball_y = mBall->get_pos_y();
saveState.ball_vel_x = mBall->get_vel_x();
saveState.ball_vel_y = mBall->get_vel_y();
saveState.ball_type = mMiddleMenu->get_selected_option();

strncpy(saveState.player1_name, player1->get_user_name().c_str(), 19);
saveState.player1_name[19] = '\0'; // Ensuring that the name ends to \0
that is standar for string types
strncpy(saveState.player2_name, player2->get_user_name().c_str(), 19);
saveState.player2_name[19] = '\0';

if (Saving::save_game(saveState))
{
  SDL_Log("Game saved successfully");
  mMenu->set_saved_file_exists();
  mNoticeMenu->set_notice_id(GAME_SAVED);
  mGameState = game_state::Notice_Menu; // We go back to the main menu
}
else
{
   SDL_Log("Failed to save game");
   mIsRunning = false;
}
```

> La sauvegarde du jeu est réalisée dans un fichier nommé `game_pong-save_849374.txt`. Ce fichier reste disponible jusqu'à ce que le joueur reprenne la partie sauvegardée ou choisisse de démarrer une nouvelle partie, auquel cas il sera automatiquement supprimé. Ainsi, votre progression est préservée même après avoir quitté le jeu.

Game retrieve logic

```
SaveState savedState;
if (Saving::load_game(savedState))
{
    player1->set_user_score(savedState.score1);
    player2->set_user_score(savedState.score2);

    player1->set_user_name(savedState.player1_name);
    player2->set_user_name(savedState.player2_name);

    racket1->set_pos_y(savedState.paddle1_y);
    racket2->set_pos_y(savedState.paddle2_y);
    ball_creation(savedState.ball_type);
    mBall->set_position(savedState.ball_x, savedState.ball_y);
    mBall->set_velocity(savedState.ball_vel_x, savedState.ball_vel_y);
    update_background_color();

    Saving::delete_save(); // Delete the saved game file once we have
loaded the game state

    mGameState = game_state::Playing;

    SoundEffects::change_music_track(mBackgroundMusic);
}
```

Choisir le type de la balle [Dounia]

Par défaut, la balle du Pong est de forme circulaire, mais pourquoi ne pas la personnaliser ? À chaque début de partie, vous pouvez sélectionner l'une des 3 formes proposées :

1. Cercle : avec une image graphique pour la détection de collision [SDL forme utilisée pour détecter les collisions].
2. Triangle
3. Carré

Ce n'est qu'une preuve de concept : rien ne vous empêche d'imaginer et d'intégrer des formes plus originales dans l'interface graphique.

Changement de la musique [Yanis]

Grâce à la bibliothèque SDL Mixer, nous pouvons gérer différents effets sonores et musiques avec des fonctions de fade-in et fade-out. Chaque mode peut ainsi avoir sa propre ambiance sonore, pour rendre l'expérience de jeu encore plus immersive.

Voici l'implémentation:

```
void SoundEffects::change_music_track(Mix_Music *music_file,
                                      int fade_out_duration,
```

```
                                        int fade_in_duration,
                                        int volume)
{
    Mix_FadeOutMusic(fade_out_duration);
    // SDL_Delay(5);
    Mix_FadeInMusic(music_file, -1, fade_in_duration);
    Mix_VolumeMusic(volume);
}
```

Chiffrement des données [Vasilis]

La sauvegarde des données utilise un système de chiffrement XOR simple avec une clé rotative:

```
class SavingEncryption {
private:
    static const std::vector<uint8_t> KEY;

    static void encryptData(std::vector<uint8_t>& data) {
        for (size_t i = 0; i < data.size(); ++i) {
            data[i] ^= KEY[i % KEY.size()];
        }
    }
};
```

Les données sont chiffrées avant l'écriture sur le disque et déchiffrées lors de la lecture, assurant une protection basique des sauvegardes.

> Inspiré de https://www.101computing.net/xor-encryption-algorithm/ L'utilisation de XOR permet à la même opération de chiffrer et de déchiffrer

---

Les objets [1. Yanis, 2. Dounia, 3. Vasilis] - Diapo 5

Dans ce projet, toutes les fonctionnalités ont été implémentées sous la forme d'objets, garantissant ainsi la modularité, la flexibilité et une organisation claire du code. Chaque élément du jeu Pong est représenté par une classe spécifique, ce qui permet une maintenance aisée et une évolutivité simplifiée du programme.

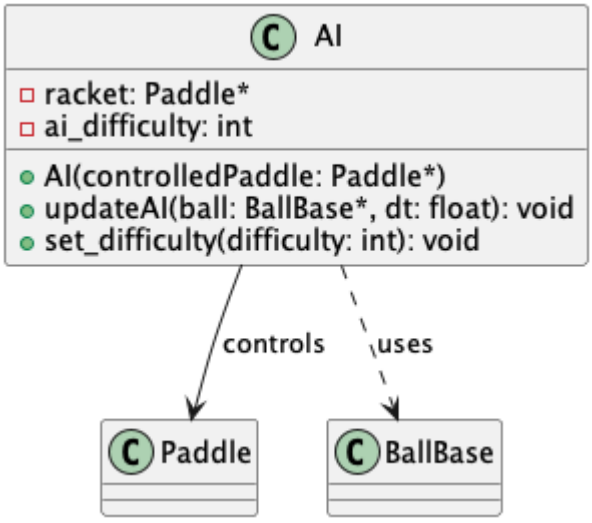
Voici les classes essentiels que nous avons définies :

Class	Description	Fichier
AI	Intelligence artificielle pour contrôler une raquette automatiquement	
BallBase	Classe de base abstraite pour tous les types de balles dans le jeu car nous proposons différents types de balles à choisir avant de lancer le jeu	
ClassicBall	Implémentation classique de balle circulaire héritant de BallBase	
SquareBall	Implémentation de la balle en forme de carré héritant de BallBase	

Class	Description	Fichier
TriangleBall	Implémentation de la balle en forme de triangle héritant de BallBase	
InvisiblePower	Rend la balle temporairement invisible. Il hérite de la classe Power	
Power	Représente les éléments de power-up qui affectent le gameplay comme le changement de la taille de la raquette, ou rendre la balle invisible	
Letter	Représente une lettre dans le mode de jeu Storytime. Contient toute la fonctionnalité pour gérer les mots dans ce mode Storytime	
Game	Contient tous les paramètres principaux, surtout les références de tous les autres objets mentionnés dans cette liste	
Paddle	Représente une raquette (paddle) de joueur	
User	Représente un joueur dans le jeu avec son nom et le suivi du score	

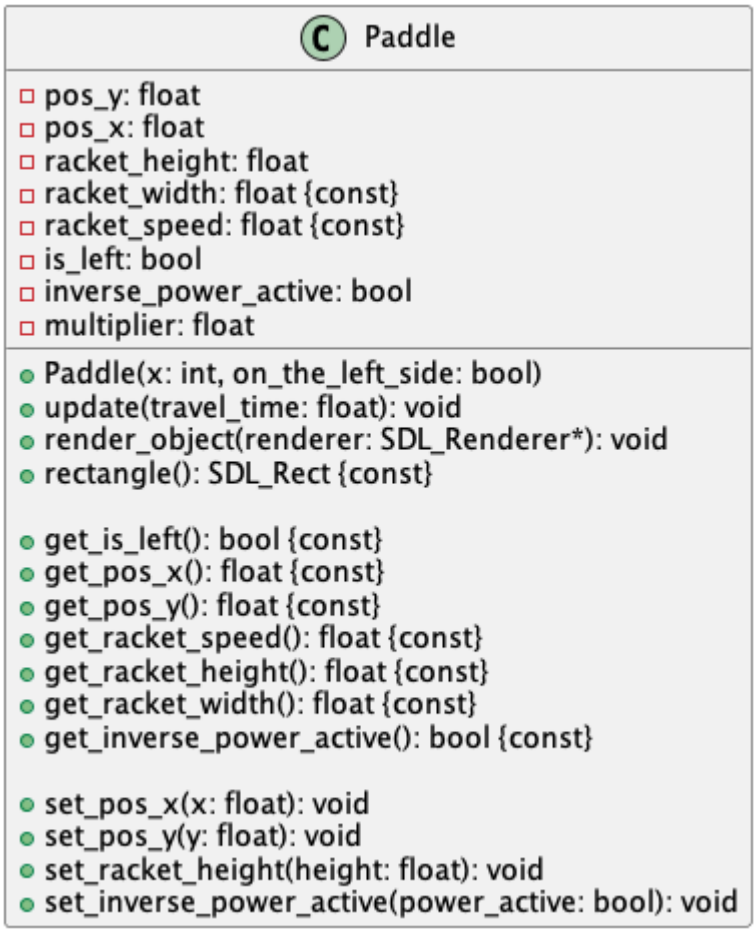
Les graphes UML pour les utiliser dans les diapos:

Pour mieux structurer notre projet et assurer une architecture claire et maintenable, nous avons modélisé les principales classes du jeu sous forme de diagrammes UML. Ces diagrammes UML permettent de visualiser l'architecture du projet et les interactions entre les classes. Cette structuration facilite la compréhension du code, son évolutivité et sa maintenance.

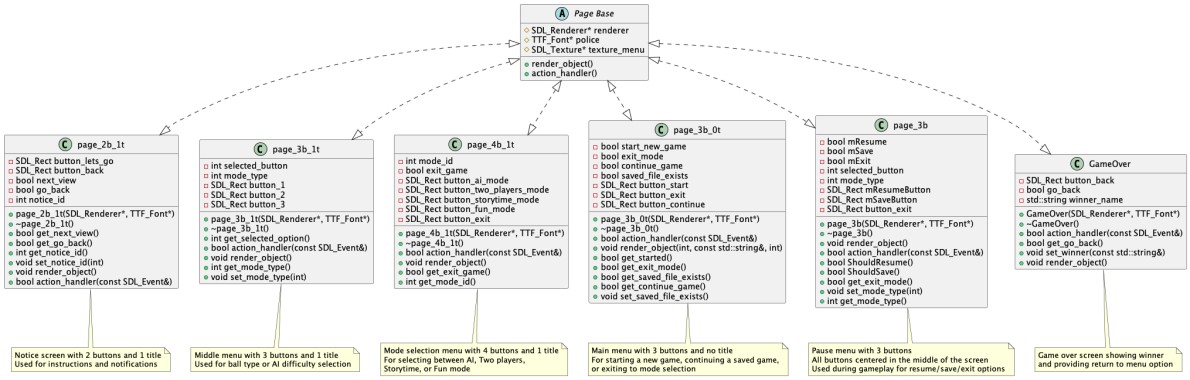
Class	UML
AI	

Class UML

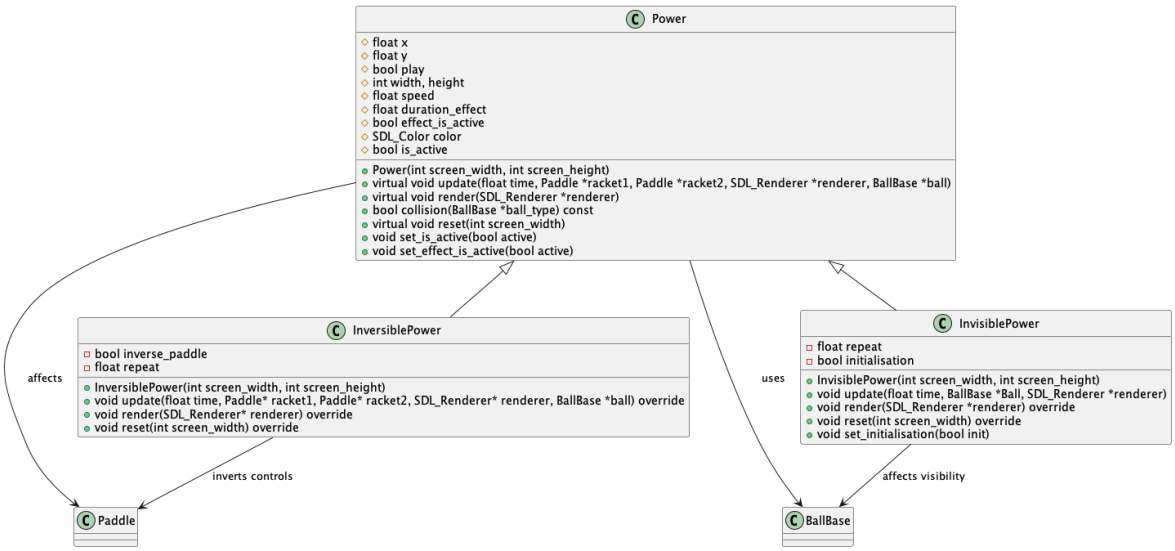
Paddle

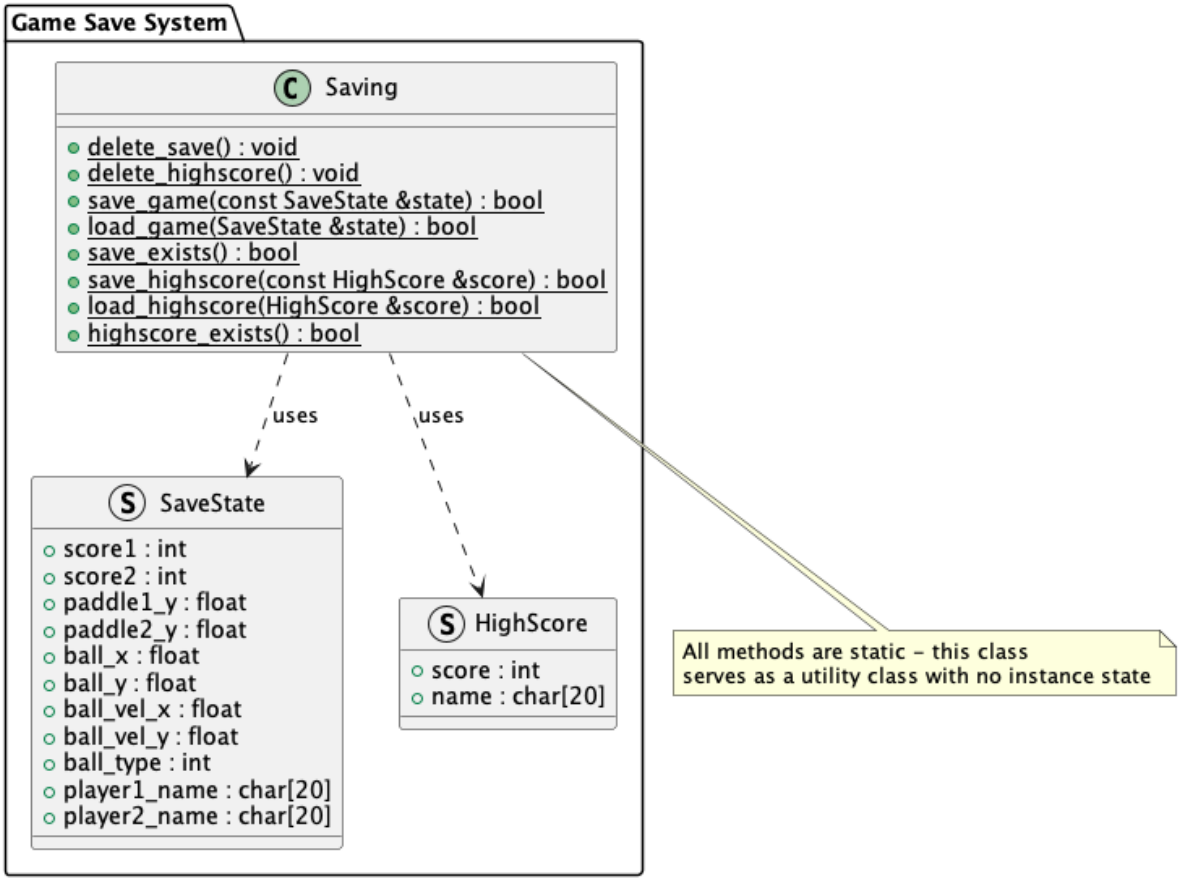
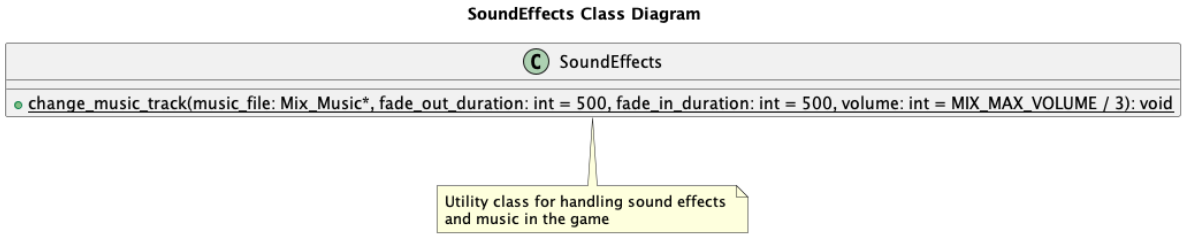
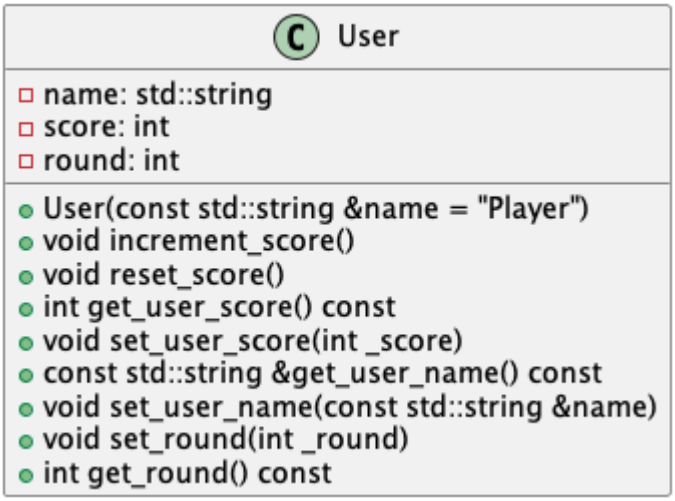


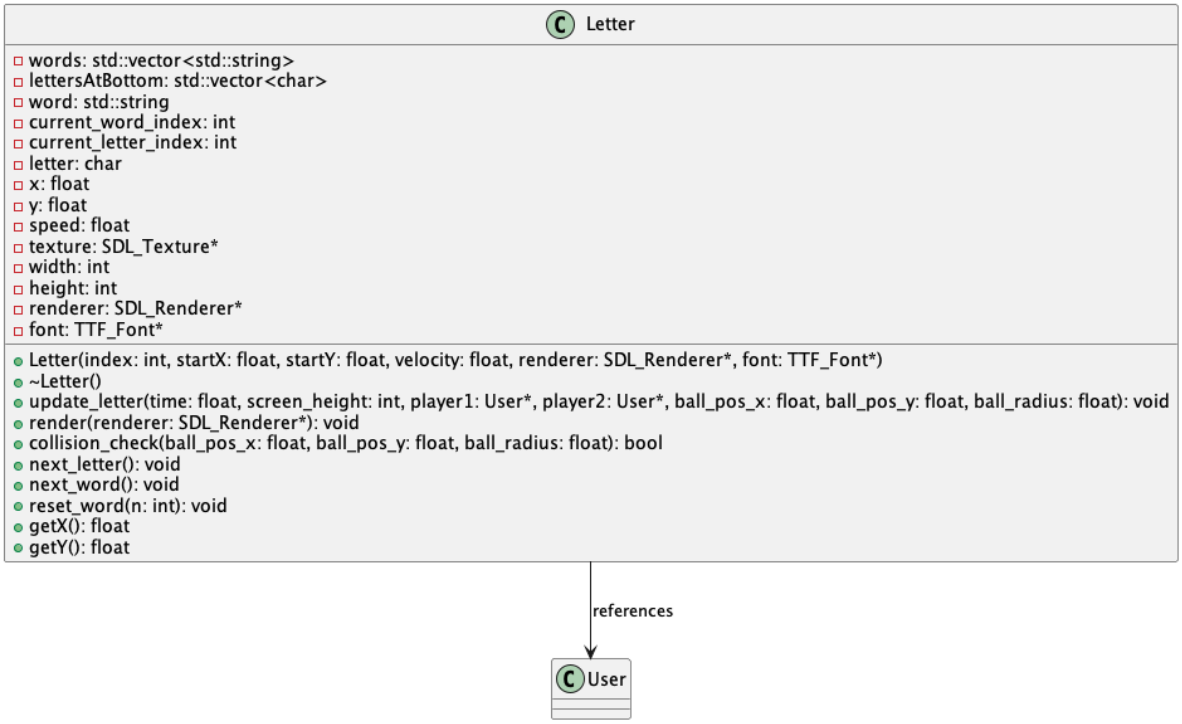
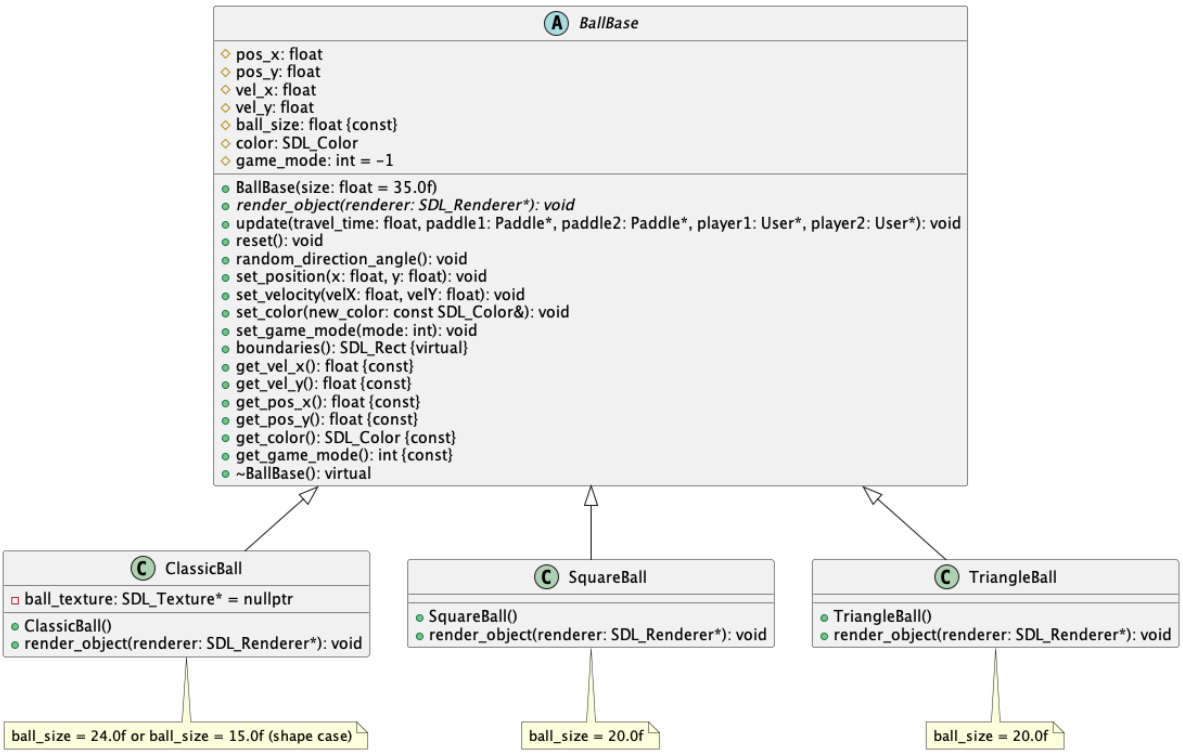
Pages (all)



Powers (all)



Class	UML
Save	
Sound	
User	

Class	UML
Letter	
Balls	

Avec cette approche, chaque élément du jeu remplit un rôle bien défini et reste modulaire, ce qui permet d'ajouter de nouvelles fonctionnalités (comme le mode réseau) sans perturber l'ensemble du projet. Nous avons créé un diagramme UML pour les différentes dépendances entre les classes. De plus, les différentes notions de programmation utilisées dans chaque partie y sont indiquées.

Les concepts clés - Diapo 6

L'interface graphique [Yanis]

Maintenant que nous avons une vue d'ensemble des différentes pages et des éléments interactifs du jeu, intéressons-nous à la façon dont l'interface graphique est conçue et gérée.

Nous utilisons SDL pour afficher et rendre toutes les formes et objets du jeu dans une fenêtre aux dimensions prédéfinies dans le fichier `macros.hpp` (plus de détails dans la section ***Pourquoi macros.hpp***).

Le programme principal repose sur la classe `Game` , qui orchestre l'ensemble du jeu à travers trois méthodes clés :

1. `initialise()` – Initialise tous les paramètres et variables nécessaires au jeu.
2. `loop()`– Gère la boucle principale du jeu.
3. `close()`– Libère les ressources et termine proprement l'exécution.

La méthode `loop()` constitue le cœur du jeu : il s'agit d'une boucle while qui tourne en continu tant que le jeu est actif. Cette boucle s'arrête uniquement si la variable booléenne `mIsRunning` est définie sur `false`, soit lorsque le joueur ferme la fenêtre SDL, soit lorsqu'il sélectionne "Exit Game".

```
void Game::loop()
{
    while (mIsRunning) // set to false when we either tap on the X to
close the SDL window or when we tap on the Exit game button
    {
        game_logic();
        game();
        output();
    }
}
```

Loop

Dans cette boucle, trois fonctions essentielles assurent le bon déroulement du jeu :

- `game_logic()` : Gère la logique principale et décide des transitions entre les pages, menus et événements du jeu.
- `game()` : Met à jour l'état du jeu en fonction des actions du joueur, détermine si une partie est terminée et applique les règles.
- `output()` : Génère et affiche les éléments visuels sur l'interface SDL en fonction des paramètres définis par la logique du jeu.

Ces trois fonctions fonctionnent en synergie pour offrir une expérience fluide et dynamique, assurant que le jeu réagit de manière cohérente aux interactions du joueur.

Héritage + Def [Vasilis]

L'héritage est largement utilisé pour étendre la fonctionnalité des classes de base. Les trois types de balles (`ClassicBall`, `SquareBall` et `TriangleBall`) héritent tous de la classe abstraite `BallBase`. Par exemple, dans `classic_ball.hpp`, nous voyons :

```
class ClassicBall : public BallBase {
public:
    ClassicBall() : BallBase(24.0f) {}
    void render_object(SDL_Renderer *renderer) override;
    // ...
};
```

Dans le domaine des power-ups, nous avons également une hiérarchie d'héritage. Les classes `InvisiblePower` et `InversePower` héritent de la classe `Power`, comme on peut le voir dans `invisible_power.hpp` et `inverse_power.hpp`. Cela permet de partager le comportement commun tout en spécialisant certaines fonctionnalités

Polymorphisme + Def [Yanis]

Le polymorphisme est implémenté à travers l'utilisation de méthodes virtuelles et leur redéfinition dans les classes dérivées. Un exemple clair se trouve dans la hiérarchie des balles, où la méthode `render_object()` est définie différemment dans chaque type de balle :

- Dans `classic_ball.cpp`, elle dessine un cercle.
- Dans `square_ball.cpp`, elle dessine un carré.
- Dans `triangle_ball.cpp`, elle dessine un triangle.

Le jeu peut manipuler n'importe quel objet dérivé de `BallBase` de manière uniforme, en appelant `mBall->render_object(renderer)` dans `game.cpp`, sans se soucier du type spécifique de balle utilisé.

De même, les power-ups démontrent le polymorphisme avec leurs méthodes `update()` et `render()` qui sont appelées de manière générique mais exécutent un comportement spécifique à chaque type de power-up.

Abstraction + Def [Dounia]

L'abstraction est implémentée principalement à travers les classes abstraites du projet. La classe `BallBase` est un excellent exemple d'abstraction. Dans le fichier `ball_base.hpp`, nous définissons une interface commune pour tous les types de balles avec des méthodes abstraites comme `render_object()`. Cette méthode est déclarée virtuelle pure (`= 0`), obligeant toutes les classes dérivées à fournir leur propre implémentation.

```
class BallBase {
public:
    // ...
    virtual void render_object(SDL_Renderer *renderer) = 0;
    // ...
};
```

De même, la classe `Power` dans `power.hpp` fournit une abstraction pour les différents types de power-ups du jeu, avec des méthodes virtuelles qui peuvent être redéfinies par les classes dérivées comme `InvisiblePower` et `InversePower`.

Fonctions lambda + Explication c'est quoi en relaite (def suffit pas) [Dounia]

Nous utilisons des fonctions lambda pour contrôler les limites physiques de la raquette (paddle). Cette approche nous permet d'obtenir un code modulaire, facilitant la mise à jour des fonctionnalités liées au déplacement et aux contraintes de position de la raquette.

Les lambdas sont particulièrement adaptées à notre cas, car elles nous permettent de définir des fonctions anonymes tout en bénéficiant d'un typage automatique, simplifiant ainsi l'écriture du code.

Les deux fonctions lambda utilisées sont:

- `auto move_paddle = [this](float delta, float time)` utiliser pour bouger l'objet paddle (la raquette)
- `auto adjust_boundaries = [this]()` responsable de vérifier et ajuster les limites de la raquette

```
auto move_paddle = [this](float delta, float time)
    {
        this->set_pos_y(this->get_pos_y() + delta * this-
>get_racket_speed() * time);
    };


auto adjust_boundaries = [this]()
    {
        if (this->get_pos_y() < this->get_racket_height() / 2.0f)
        {
            this->set_pos_y(this->get_racket_height() / 2.0f);
        }
        else if (this->get_pos_y() > 600.0f - this->get_racket_height() /
2.0f)
        {
            this->set_pos_y(600.0f - this->get_racket_height() / 2.0f);
        }
    };
```

Encapsulation + Def [Vasilis]

L'encapsulation est présente dans presque toutes les classes du projet, avec une distinction claire entre les interfaces publiques et les détails d'implémentation privés. Par exemple, la classe `User` dans `user.hpp` encapsule les données relatives au joueur :

```
class User {
private:
    std::string name;  // Le nom de l'utilisateur
    int score;         // Le score actuel de l'utilisateur
    int round;         // Le numéro de round actuel

public:
```

```
    // Méthodes pour interagir avec les données encapsulées
    void increment_score();
    void reset_score();
    int get_user_score() const;
    // ...
};
```

Un autre exemple d'encapsulation se trouve dans le fichier `game_save.cpp`, où un namespace anonyme est utilisé. Tous les détails sont disponibles ci-dessous.

Namespace

Dans le fichier game_save.cpp, nous utilisons un namespace anonyme afin d'encapsuler les constantes sensibles (comme la clé de chiffrement XOR) ainsi que les fonctions utilitaires dédiées au codage et au décodage des données de sauvegarde.

Ce choix présente plusieurs avantages :

1. **Sécurité renforcée** : empêche tout accès externe aux mécanismes internes du système de sauvegarde.
2. **Prévention des conflits** : réduit les risques de collision de noms avec d'autres parties du code.
3. **Organisation claire** : regroupe logiquement les éléments qui interagissent ensemble, améliorant ainsi la lisibilité et la maintenabilité du code.

L'utilisation d'un namespace anonyme garantit donc une encapsulation stricte et protège les données critiques du jeu contre toute manipulation involontaire ou non autorisée. Cette approche garantit que ces éléments ne sont accessibles que depuis ce fichier, renforçant ainsi la sécurité du mécanisme de sauvegarde.

```
namespace
{
    unsigned char codec_byte(unsigned char byte)
    {
        return byte ^ KEY; // XOR operation
    }

    void codec_float(float &value)
    {
        unsigned char *bytes = reinterpret_cast<unsigned char *>(&value);
// casting to return a float value to byte representation
        for (size_t i = 0; i < sizeof(float); ++i)
        {
            bytes[i] = codec_byte(bytes[i]);
        }
    }

    void codec_int(int &value)
    {
        unsigned char *bytes = reinterpret_cast<unsigned char *>(&value);
        for (size_t i = 0; i < sizeof(int); ++i)
```

```
        {
            bytes[i] = codec_byte(bytes[i]);
        }
    }

    void codec_string(char *str, size_t length)
    {
        for (size_t i = 0; i < length; ++i)
        {
            str[i] = codec_byte(str[i]);
        }
    }
}
```

Cette organisation du code illustre parfaitement le principe d'encapsulation, un pilier fondamental de la programmation orientée objet. En limitant l'accès aux éléments internes du système de sauvegarde, nous renforçons la sécurité, l'isolation et la robustesse globale du jeu.

> Afin de réduire la longueur du rapport, nous avons retiré les commentaires détaillés des différentes fonctions. Pour une explication complète et une vue d'ensemble du code, vous pouvez consulter directement game_save.cpp.

Les foncteurs [Vasilis]

Foncteur	Descriptions	Fichier
triangle_renderer	Foncteur pour le rendu des formes triangulaires	renderers.cpp
square_renderer	Foncteur de rendu de formes carrées	renderers.cpp

L'utilisation de foncteurs nous permet d'ajouter facilement de nouveaux types de formes et de les tester individuellement.

> Cette approche nous a permis d'accélérer le développement en permettant des tests isolés des différents SDL renderers.

Problemes recnontres [Dounia] - Diapo 7

Voici les differents problemes / chalenges reconnus:

- segmentation fault when we declated the "new Power(..." due to an issue on the constructror => pas d'allocation memoire pour l'objet en question, donc tous les operations tels que power.update(... n'etait pas succesives.
- Pendant le development de la fonctionalite HighScore, au debut nous avons pensé que pour etre capable de trouer le highscore entre differents sessions du jeu, il faudrait sauvegarder tous les meilleurs scores de chaque jeu. En fait ca suffit, d'avoir seulemnt l'information du derniere highscore. Comme ca nous evirons de chercher sur un fichier qui augment de plus en plus
- Pendant le development de notre fonctionalite pour la sauvegarde, il faudrait trouver une maniere que le fichier de la sauvegarde soit etre lu et dedruit en "bonnes endroits". En mode, la structure la plus

optimale est la suivnate: si jamais un fichier de la sauvegarde existe, alors l'utilisateut peut chosir de continuer par ce fichier. S'il chosiit cette option, on va recuperer les donnes et on va supprimer le fichier directement et le jeu va se lancer. L'autre scenario est que l'utilisateur va choisir de commencer un nouveau jeu. Dans ce cas là, si il existe un fichier de sauvegarder, et on va lancer le nouveau jeu. - On prefere de faire ca, que de ecrire en desus deu fichier existant pour eviter des erreurs des informations qui restent dans le fichier qui chifré

- Quand on etait en mode storytime ou fun, nous avons constaté que à la fin de chaque jeu avec de tours, si on voudrais recomemncer un jeu de ces deux modes, on ete rediregie directement sur page game over parce que la logique de tous n'etait complet. Effectiveemnt au debut de chaque jeu, on reinitialise les indicateurs iimportante stels que: type de balle, infos des joueurs, et le type du jeu, mais nous avons oublié de reinitialiser les informations concernant les tours.
- Pour detcter les collisions entre la balle et les differents composantes tels que les robots ou les lettres, au debut on etait basé sur la detection de la supperposition des differrentes pixels entre les differents objets, donc il faudrait declarer differents fonctions pour les differentes types de balles, tandis que tout ca est observable via SDL rendering. SDL viens avec une fonction integré SDL_HasIntersection
- Nous avons commencé le projet avec un Makefile, sauf que pas tout le monde a le memem ordinateur et le meme liens symboliques pour acceder auxx bilbiotheques de dependance de base. Donc, quand on essaye de tounrer le jeu sur un linux, il faudrait modifier tous les differents chemins. Le CMake nous aidé pour resoudre ce soucis. Nous avons declaré les chemins pur les differents architectures dans un fichier CmakeLists.txt. En combinant notre script d'installation de base, nous avons une integration parfait qui permet d'installer tous les librairies et les verfier avant creer le makefile via Cmake et toutner le logiciel

---

Pour aller plus loin [Vasilis] - Diapo 8

Initialement, nous avons tenté d'implémenter un mode multijoueur en réseau via TCP avec une architecture client-serveur. Cependant, nous nous sommes rapidement heurtés à la complexité de cette intégration.

En effet, cette fonctionnalité aurait dû être pensée dès le début du projet afin d'être intégrée naturellement dans l'architecture existante. L'ajout tardif d'un mode réseau implique de lourdes modifications sur la structure actuelle du code, ce qui s'avère être un défi technique conséquent.

Malgré ces difficultés, nous avons commencé le développement de cette partie dans les fichiers `network.cpp` et `network.hpp`, en nous concentrant sur les aspects suivants :

- Le contrôle des raquettes à distance
- La réception et la synchronisation des positions des différents éléments (balle, raquettes, etc.)

Conclusion [Yanis] - Diapo 9

Pong, mais en mieux ! Notre projet revisite ce grand classique du jeu vidéo en exploitant pleinement les principes de la programmation orientée objet, nous permettant de créer un code modulaire, extensible et maintenable.

Grâce à l'abstraction et à l'héritage, nous avons structuré notre jeu avec des interfaces claires et des hiérarchies logiques. Le polymorphisme nous a permis de manipuler différents objets de manière uniforme,

tandis que l'encapsulation a assuré la protection et l'intégrité des données. Nous avons également tiré parti des foncteurs et des fonctions lambda pour encapsuler des comportements spécifiques, rendant notre implémentation plus souple et efficace.

Mais ce projet ne se limite pas à un simple exercice de programmation ! Nous avons voulu pousser l'expérience plus loin, en intégrant plusieurs modes de jeu inédits, un système de sauvegarde sécurisé, une interface graphique fluide avec SDL, et même une tentative d'implémentation du multijoueur en réseau.

Le résultat ? 🎾 Un jeu fun, dynamique et personnalisable, qui vous permet de revivre l'expérience du Pong… mais avec une touche de modernité !

👉 **Prêt à relever le défi et à battre le high score ? Jouez, et montrez-nous qui est le véritable maître du Pong !**

PENDANT LA DEMO

Compilation [pendant la demo : Vasilis]

Nous avons intégré un fichier **CMakeLists.txt** afin de faciliter la compilation du projet sur les principaux systèmes d'exploitation tels que macOS et Linux. Toutefois, l'interface graphique nécessite plusieurs dépendances spécifiques.

Pour simplifier cette étape, nous avons également créé un script Bash qui vérifie automatiquement si toutes les dépendances sont installées. Si ce n'est pas le cas, il se chargera de télécharger et d'installer ce qu'il manque. Vous trouverez la liste exhaustive de ces dépendances dans la section **Dépendances** .

Nota bene

> Sur Linux, le script télécharge et installe automatiquement les bibliothèques SDL nécessaires. Toutefois, si l'une d'entre elles requiert une autre dépendance spécifique, vous devrez l'installer manuellement. Une fois la dépendance installée, relancez simplement le script avec la commande `bash play.sh`.
>
> Pour aller plus loin: d'abord, la commande `mkdir -p build` crée un répertoire isolé pour les fichiers générés pendant la compilation si il n'existe déjà, puis `cd build` nous positionne dans ce dossier, suivie de `cmake ..` qui analyse le fichier CMakeLists.txt du projet pour configurer l'environnement et détecter les bibliothèques nécessaires. Enfin `cmake --build .` lance la compilation effective du code source.

Instructions de lancement

Pour démarrer le programme en mode automatique, suivez les étapes suivantes :

1. Ouvrez un terminal et se rediriger vers un repertoire souhaité (par exemple `cd ~/Downloads`)
2. Faire un clone du projet via la commande : `git clone https://github.com/vskarleas/The-New-Pong`
3. Acceder au projet cloné via la commande : `cd The-New-Pong`
4. Saisissez `chmod 777 play.sh` dans le terminal, puis lancez le script avec `bash play.sh`.

Dépendances

Voici la liste des dépendances indispensables au bon fonctionnement du programme :

- **SDL2** : Bibliothèque principale pour la gestion de la fenêtre et des événements
- **SDL2_ttf** : Bibliothèque pour le rendu du texte (polices TrueType)
- **SDL2_mixer** : Bibliothèque pour la gestion du son et de la musique
- **SDL2_image** : Bibliothèque pour le chargement d'images (formats multiples)
- **SDL2_net** : Bibliothèque pour les fonctionnalités réseau

Pourquoi macros.hpp

Le fichier macros.hpp joue un rôle central dans notre projet en servant de référentiel unique pour toutes les constantes globales du jeu. Il permet de centraliser et de faciliter la gestion des paramètres essentiels, tels que :

- Les dimensions de la fenêtre du jeu
- Les identifiants des modes de jeu (ex. : mode IA, mode 2 joueurs)
- Les constantes associées à la navigation dans les menus
- Les niveaux de difficulté

Grâce à ce fichier, nous avons assuré une meilleure lisibilité et une maintenance simplifiée, en évitant la dispersion des constantes dans l'ensemble du code.