Projet Multijoueur en C++

Auteurs: Dounia Bakalem, Yanis Sadoun, Vasileios Filippos Skarleas

Le projet

Dans le cadre du projet en C++ pour ce semestre, nous étions demandés de créer un jeu. Le but principal est de mettre en œuvre les différentes notions que nous avons vues en cours. Plus particulièrement:

- L'héritage
- Le polymorphisme
- L'abstraction
- Les foncteurs
- Les fonctions lambdas
- CMake
- Les fonctions virtuelles

Le jeu

Nous avons décidé de développer et implémenter le Pong, un des premiers jeux vidéo d'arcade et le premier jeu vidéo d'arcade de sport. Cependant nous souhaitons ajouter des fonctionnalités et le personnaliser pour qu'il devienne encore plus intéressant que la version originale du jeu. C'est pourquoi nous avons introduit 4 différents modes de jeu:

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

Nous avons inclus des instructions dans le jeu avec tous les détails que vous allez trouver dans la partie **Les différents modes**.

Compilation

Nous avons intégré un fichier CMakeLists.txt qui permet la compilation aux différents systèmes d'exploitation majeurs tels que Mac et Linux sans aucun souci. Cependant, il y a plusieurs dépendances pour que l'interface graphique fonctionne.

Donc, pour simplifier toutes ces démarches nous avons créé un script bash qui automatiquement va détecter si toutes les dépendances sont respectées. Si elles ne sont pas respectées, il va télécharger automatiquement les bibliothèques manquantes. Pour votre information, vous allez trouver toutes les dépendances du jeu dans la partie **Dépendances**.

Instructions de lancement

Pour démarrer le programme automatique, veuillez suivre les instructions d'installation automatique :

1. Faire un clone du projet
2. Ouvrez un terminal
3. Saisissez `chmod 777 play.sh` dans le terminal, puis lancez le script avec `bash play.sh`.

Notes

> Dans le cas de Linux, il télécharge sans aucun souci les bibliothèques SDL, mais si une bibliothèque a une autre dépendance, il faut que vous téléchargiez manuellement cette dépendance. Après, il suffit de relancer le script avec `bash play.sh`.

Documentation

Vous pouvez visiter le website https://pong.madebyvasilis.site qui inclut tous les détails des différentes classes, structures et fonctions.

Dépendances

Voici les dépendances du programme principal

- `SDL2` - La bibliothèque principale SDL2
- `SDL2_ttf` - Bibliothèque pour le rendu de texte (polices TrueType)
- `SDL2_mixer` - Bibliothèque pour la gestion du son/musique
- `SDL2_image` - Bibliothèque pour le chargement d'images
- `SDL2_net` - Bibliothèque réseau pour SDL2

Les différents modes

Classic

Le concept original de Pong est un simulateur simple de ping-pong. Une petite boule, se déplace à travers l'écran, rebondissant sur les rebords du haut et du bas, et le joueur commande un « paddle », qui glisse de haut en bas à travers les extrémités de l'écran. Si la boule frappe la palette, elle rebondit vers l'autre joueur. Si elle manque la palette, l'ordinateur marque un point. La boule rebondit de différentes manières selon la façon dont la boule touche la palette.

Dans ce mode vous allez trouver les fonctionnalités suivantes. Plus des informations pour chacune sont disponibles dans la section Fonctionnalités:

- High Score
- Game Save

Nous avons décidé que pour cette partie, au lieu de fixer un score maximum qui pourrait signaler la fin du jeu, dans notre version les joueurs peuvent tout simplement décider un score entre eux à l'oral et commencer à jouer. Pour arrêter il suffit de choisir "End the game".

La motivation essentielle dans ce mode est "qui va dépasser le dernier high score ?".

AI mode

Le principe est le même comme au mode classic sauf que ici, pong ne peut être joué que par un joueur seul, avec la palette opposée commandée par l'ordinateur.

Storytime mode

Dans ce mode, les deux joueurs vont jouer 5 tours. Le gagnant est le joueur qui a gagné le plus grand nombre de tours. Chaque tour est composé de 12 parties. Le gagnant d'un tour est le joueur qui atteint 12 points en premier.

Ici, nous allons voir des lettres tomber par le haut de l'écran. Si vous ciblez et touchez les lettres, vous gagnez un point en extra.

Quand vous touchez les lettres, petit à petit, un mot va être construit et comme ça vous pouvez lire la phrase secrète.

Fun mode

Vous allez trouver les mêmes règles comme au storytime mode en ce qui concerne qui gagne. Ce mode est assez particulier car nous avons introduit des easter eggs pour les joueurs. Par exemple:

- Si quelqu'un touche le cube qui tombe, son paddle va grandir temporairement
- Si vous touchez une étoile, la balle va disparaître pour une période de temps courte

Fonctionnalités

High Score

Cette fonctionnalité est disponible seulement en mode classique du jeu. La logique du jeu vérifie à chaque instant si jamais quelqu'un arrive à faire un score supérieur au max qui est sauvegardé avant.

La sauvegarde est faite sur un fichier nommé game_pong-highscore_849216.txt qui est chiffré, donc c'est impossible pour quelqu'un de modifier le contenu s'il ne connaît pas comment il est chiffré. Les informations qui sont stockées sont bien évidemment le dernier high score et le nom du joueur.

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

Est-ce que vous souhaitez faire une pause, et continuer plus tard pour essayer de dépasser le dernier high score ?

Vous avez l'option de sauvegarder l'état de votre jeu et revenir quand vous le souhaitez. Comme attendu, le chiffrement est pris en compte dans cette fonctionnalité aussi.

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 est faite aussi sur un fichier qui est stocké sous le nom game_pong-save_849374.txt. Même si on ferme complètement le jeu, le fichier reste sur le répertoire. Il est supprimé si et seulement si quelqu'un vient de continuer par le jeu qui était sauvegardé ou s'il choisit de commencer un nouveau 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

Par défaut, le pong vient avec la balle classique circulaire, mais si quelqu'un souhaite un peu de personnalisation, nous offrons cette option.

Au début de chaque mode, vous êtes demandé quel est le type de la balle que vous souhaitez. Nous avons intégré 3 formes basiques:

- cercle avec une image graphique [SDL forme utilisée pour détecter les collisions]
- triangle
- carré

Il s'agit tout simplement d'une preuve de concept qui nous indique que nous sommes libres de designer et intégrer tout ce qu'on souhaite dans l'interface graphique.

Changement de la musique

Nous avons réussi à implémenter correctement la bibliothèque de SDL Mixer qui nous permet de fade-in et fade-out différents sons (musique) et différents effets de son.

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

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

Dans le cadre ce projet en programmation orientée objets, nous avons créé des objets qui représentent des éléments du jeu pong. Il faut noter quand même que toutes les fonctionnalités sont codées comme si c'était des objets, pour assurer la modularité du code et la bonne organisation des fichiers. Voici les différentes classes que nous avons créées:

Class	Description	Fichier
AI	Intelligence artificielle pour contrôler une raquette automatiquement	ai.cpp
BallBase	Classe de base abstraite pour tous les types de balles dans le jeu car nous proposons différents types de balles à choisir avant lancer le jeu	ball_base.pp
ClassicBall	Implémentation classique de balle circulaire héritant de BallBase	classic_ball.cpp
Game	Il contient tous les paramètres principaux, surtout les références de tous les autres objets mentionnés dans cette liste	game.cpp
GameOver	Gère l'écran de fin de partie lorsqu'une partie est terminée ou si on choisit de terminer manuellement une partie	game_over.cpp

Class	Description	Fichier
GUI	Classe utilitaire fournissant des fonctionnalités d'interface utilisateur (donner notre prénom via SDL))	gui.cpp
HighScore [structure]	Structure représentant un record de score élevé. Il gère la sauvegarde de ces données spécifiques	game_save.cpp
InvisiblePower	Rend la balle temporairement invisible. Il hérite de la classe Power	invisible_power.cpp
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	power.cpp
Letter	Représente une lettre dans le mode de jeu Storytime. Contient toute la fonctionnalité pour gérer les mots dans ce mode Storytime	letter.cpp
Paddle	Représente une raquette (paddle) de joueur	paddle.cpp
SaveState [structure]	Structure représentant l'état complet du jeu pour la sauvegarde/le chargement	game_save.cpp
Saving	Classe utilitaire de sauvegarde pour gérer la sauvegarde de la partie et la fonctionnalité de score élevé	game_save.cpp
SoundEffects	Classe pour gérer les effets sonores et la musique dans le jeu	sound_effects.cpp
SquareBall	Implémentation de la balle en forme de carré héritant de BallBase	square_ball.cpp
TriangleBall	Implémentation de la balle en forme de triangle héritant de BallBase	triangle_ball.cpp
User	Représente un joueur dans le jeu avec son nom et le suivi du score	user.cpp
page_2b_1t	Écran d'avis avec 2 boutons et 1 titre	page_2b_1t.cpp
page_3b	Menu de pause avec 3 boutons	page_3b.cpp
page_3b_0t	Classe de menu principal avec 3 boutons et aucun titre	page_3b_0t.cpp
page_3b_1t	Classe de menu intermédiaire avec 3 boutons et 1 titre	page_3b_1t.cpp
page_4b_1t	Définit le menu de sélection de mode avec 4 boutons et 1 titre	page_4b_1t.cpp

Les foncteurs

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.

Structure des pages

Ce jeu est fourni avec des structures de page prédéfinies spécifiques que nous utilisons pour restituer différents contenus de manière dynamique ou non. Voici les spécifications de ces fichiers :

- page_3b_1t : Il y a trois boutons au centre et un titre en gras en haut (menu du milieu précédent)
- page_2b_1t : Il y a deux boutons et une grande partie pour le texte long (avis précédents)
- page_4b_1t : Il y a quatre boutons et 1 titre en gras (menu mode précédent)
- page_3b_0t : Il y a trois boutons, les deux en haut de la page et le troisième près de la fin (menu précédent)
- page_3b : Il y a 3 boutons, tous centrés au milieu de l'écran (menu pause précédent)

La logique du jeu

L'interface graphique

Maintenant que nous avons une image plus claire concernant les différentes pages qui contiennent des éléments pour interagir avec le logiciel, nous allons analyser comment l'interface graphique est réalisée.

Nous utilisons SDL pour rendre et afficher toutes les différentes formes dans une fenêtre avec des dimensions prédéfinies dans le fichier macros.hpp (plus d'information sur la partie **Pourquoi macros.hpp**). Le programme principal, demande la création de la classe `game` qui inclut les trois méthodes suivantes:

1. Initialise
2. Loop
3. Close

Les méthodes `initialise` et `close` permettent l'initialisation de tous les différents paramètres et variables tandis que loop tourne en permanence. La méthode loop est une boucle while avec un booléen qui va arrêter de se répéter si jamais le booléen est mis à false par la logique, comme on peut constater ci-dessous:

```
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 alors, il y a la logique du jeu `game_logic()` qui est responsable de nous rediriger vers les différentes pages, messages et menus. En plus, nous allons trouver, le `game()` qui est la mise à jour des différentes états selon lesquelles on décide par exemple si une partie est terminée ou pas, et il y a aussi `output()` lequel selon les paramètres qui sont fixés par la logique, va nous rendre les objets visuels sur l'interface SDL.

Héritage

...

Polymorphisme

...

Abstraction

...

Fonctions lambda

Nous utilisons des fonctions lambda pour vérifier les limites physiques de l'objet paddle. Cette approche nous permet d'avoir un code qui est modulaire et qui nous permettra de mettre à jour la fonctionnalité de changement de position et les conditions de l'objet paddle facilement.

Plus précisément, dans notre cas les fonctions lambda étaient parfaites car elles nous permettent de définir un objet de fonction anonyme tout en gérant le typage automatiquement pour nous.

Vous allez trouver deux fonctions:

- `auto move_paddle = [this](float delta, float time)` responsable pour bouger l'objet paddle (la raquette)
- `auto adjust_boundaries = [this]()` responsable pour 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

Namespace

Nous utilisons un namespace anonyme dans le fichier game_save.cpp. Ce mécanisme permet d'encapsuler les constantes sensibles (comme la clé de chiffrement XOR) et les fonctions utilitaires de codage/décodage dans une portée strictement limitée au fichier d'implémentation. Voici les différents avantages de ce choix:

1. Le namespace empêche l'accès externe aux mécanismes de sécurité du système de sauvegarde
2. réduit les risques de collision de noms avec d'autres parties du code
3. structure logiquement les éléments qui travaillent ensemble.

```
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 incarne le principe d'encapsulation, fondamental en programmation orientée objet, tout en contribuant à la robustesse globale du système de sauvegarde du jeu.

> Pour des raisons de diminution de la longueur du rapport, nous avons enlevé les commentaires des différentes fonctions. Pour voir toutes les fonctions en détail visitez game_save.cpp

Autres cas

Tout au long du projet, nous faisons un choix si on va inclure des variables comme `private` ou `publique` aux différentes classes. En plus, nous faisons attention à la bonne utilisation de mots clés de l'encapsulation tels que `static` et `virtual`.

Pourquoi macros.hpp

Le fichier macros.hpp joue un rôle fondamental dans notre projet en servant de point central pour toutes les constantes du jeu. À travers ce fichier, nous définissons les dimensions de la fenêtre de jeu, les identifiants des différents modes de jeu (comme les modes IA ou deux joueurs), ainsi que les constantes utilisées pour la sélection des menus et les niveaux de difficulté - éléments qui nous permettent de naviguer aux différentes pages du jeu.

Pour aller plus loin

...

Versions

Le versioning est un élément clé en programmation, assurant la cohérence des modifications et facilitant la collaboration. Il est aussi primordial pour la récupération de données en cas de perte ou corruption. Au fil du projet, nous avons créé différentes versions de notre code, chacune marquant une étape importante de son évolution. Cela nous a permis de suivre les progrès, d'intégrer de nouvelles fonctionnalités et d'effectuer des corrections de manière structurée.

- V4.2.2 Saving the demi correct views
- V4.2.3 The views logic has been completed
- V5.0.1: **Added lambda functions** on the paddle.cpp. The reason why we do this is found on the paddle.cpp file and added multiple notices support
- V5.1.1: **Functors added for the different ball shaped renders on the renderers.cpp file** (it includes also the explanation why we use functors [purposes])
- V5.1.2: Added getter and setter for the notices (it will be needed on the game's main logic to showcase the correct notices) + on Makefile added the mode_menu implementation
- V5.1.3: Prepared notices so that we can return back to the mode menu
- V5.1.4: Added logic for showing the pause button or not on the game
- V6.0.1: We added the change views functionality successfully
- V6.1.0: Added updates pages structures and logic for correct AI and balls selection to their respective modes
- V6.1.1: Changes GameState to game_state for normalisation reasons

- V6.1.2: Renamed files according to the pages structure below for clarity reasons. The classes on those files has not been updated yet
- V6.1.3: Updates class names according to the pages structures below and AI logic has been implemented. Needs to be added on the game's global logic
- V7.0.1: Added the AI on the global game's logic
- V7.1.2: Added game over page and Cmake structure. Changed some function names on the game.cpp and we added an automatic installer and handler of packages.
- V8.1.0: Added user class, started network codebase, added special effects support for game's actions and buttons click actions, updated saving state to include players names and updated the game's logic to support the definition of the player's names.
- V8.1.1: Added support for showing player's names when we play
- V9.0.1: Added high score functionality on two players mode. The game now ends when the users ask for it, so this is what makes a high score. Also added support to add user names on SDL interface
- V9.1.0: Updated the gui.cpp for better results and cohesion
- V9.2.0: Storymode added. We need to update the instructions
- V9.2.1: Removed setup.cpp and setup.hpp dependencies
- V9.2.2: Removed final dependencies of setup.cpp
- V10.8.23 : Implemented fun mode with rackets size changing and ball invisibility. Macros were extended, updated games logic to respond to the new criteria and fixed some bugs on the source code (+ 8.5 hours for it to work after the modifications of Dounia and Yanis)
- V10.8.24: Old makefile removed and gitignore updated. You will avoir à performer le cmake to compile. You can use directly the play.sh script
- V10.9.1: play.sh installer was updated
- V11.0.1: Added documentation structure. Fixed some issues on the fun mode. We need to reset the paddle height at the end of the fun mode manually if the button end game is tapped otherwise we risk to have different size paddles on other game modes
- V11.0.2: Added more comments and included inverse mode there is a segmentation fault issue
- V11.1.2: Resolved segmentation fault issue. Corrected some logic bugs and updated the documentation
- V12.0.1: Game has been completed. Some comments are missing on the inversible power file

---

TO-DO list

- ☑ Find a way to propose an initial SDL window that showcases the instructions of the game. Then when the user taps on a button "Let's go", then this SDL window closes and a new one Appears where a user chooses between the modes of the game that exist. The "Simple" one which is the current version of the game, the "Storytime" one where is the game without the saving functionality, but with the user class and the high score board, and the third one which is "Hardcore" which is the same with the "Storytime" but now there is a single user that plays against an AI on the opposite Paddle. There is also the exit game button that terminates the game. When a mode is selected this SDL window closes and we are presented with the menu window of the different game modes. (definitely there are modifications on the main.cpp for this functionality to take place)
- ☑ Add a message when the game is saved successfully
- ☑ Create a user class and use this to attribute the scores
- ☑ Remove the end of game logic when score is greater than 10 for a user. Instead use the logic of the two previous points (not actually remove, but instead develop the new game mode like

game_storytime.cpp). This new mode comes with its new menu_storytime.cpp that doesn't include the continue game functionality (as expected) but there is button that allows to see the high scores table (it opens on a new SDL window). The exit game button is transformed to exit mode button allowing to go back to the SDL window where we can choose the game mode.

- [x] Have a high scores table where if any time in the future a user has a score greater than the last maximum one, then he is added on top of the Leaderboard. The leaderboard is a file that is always present on the directory of the game and it is also encrypted.
- [x] Add the AI player
- [ ] Why there is a Float set_up method on the paddle.hpp ? Ask Yanis and Dounia