DURANDEAU Alexandre

CHOPIN Jérémy

Projet EI4-SAGI

Py-Pong guidé par webcam

Sous la direction de

M. Jean-Baptiste FASQUEL

université angers

IstiA ÉCOLE D'INGÉNIEURS

Année scolaire

2017-2018

Remerciements

Avant toute chose, nous tenons à adresser nos remerciements aux différentes personnes qui nous ont permis de réaliser ce projet.

Nous remercions tout d'abord M. Fasquel qui nous a proposé ce projet et nous a accompagnés tout au long de son déroulement. Ses conseils et avis nous ont permis de résoudre de nombreux problèmes et fournir un travail de meilleure qualité.

Nous remercions aussi toutes les personnes qui ont de près ou de loin participé à ce projet en nous donnant des conseils et des avis constructifs.

Table des matières

Introduction

Lors de ce travail effectué en quatrième année d'école d'ingénieurs, nous nous sommes intéressés particulièrement au projet se basant sur le jeu Pong. L'objectif de celui-ci a été de pouvoir jouer au Pong en se servant d'une caméra pour pouvoir contrôler l'intégralité du jeu ainsi qu'une implémentation sur un support mobile du type Raspberry.

Le jeu Pong est un jeu relativement connu car il s'agit d'un des premiers jeux vidéo au monde connu du grand public. Imaginé par l'Américain Nolan Bushnell en 1972, le principe de ce jeu est de « simuler » une partie d'un sport de raquette. Il y a donc deux joueurs, chacun contrôlant une raquette et une balle qu'ils se renvoient à tour de rôle jusqu'à ce que l'un d'entre eux n'arrive pas à la renvoyer. Originellement, le jeu est contrôlé par interaction avec un clavier ou un joystick et le joueur était dans une position statique. C'est pourquoi il est intéressant de modifier ce jeu, pour que le contrôle s'effectue via un traitement d'images par le biais d'une caméra, rendant le joueur plus dynamique car il devra se mouvoir devant celle-ci pour pouvoir déplacer les raquettes.

Ce projet nous a permis de travailler sur des technologies nouvelles par rapport aux traitements d'images et aussi d'améliorer nos connaissances dans certains domaines tels que la programmation orientée objets. L'objectif de ce rapport est de présenter le déroulement de notre projet avec ses problèmes , ses solutions, les résultats que nous avons obtenus et la façon dont nous y sommes parvenus.

Dans un premier temps, nous présenterons l'organisation que nous avons mise en place pour ce projet. À la suite de cela, nous entrerons dans les détails pour décrire le développement de l'application. Finalement, nous exposerons les différents axes d'améliorations possible pour le projet.

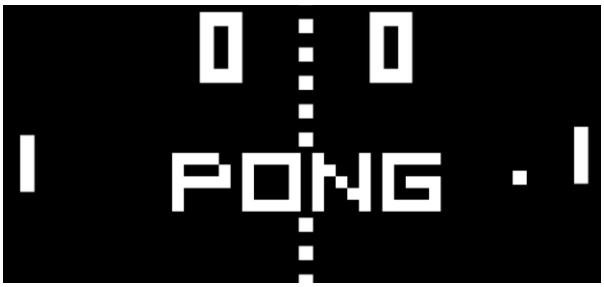


Figure 1: Ecran du jeu Pong

1) Présentation

A. Objectifs

Au commencement de ce projet, nous avions comme objectif final d'obtenir un jeu Pong contrôlé par webcam et implémenté sur une plateforme mobile Raspberry PI. Pour ce faire, nous avions comme base de travail une version du jeu Pong en itératif sous le langage Python et certaines indications sur des technologies qui pourraient nous être utiles.

Au vu des informations et du support que nous avions, il nous a été difficile de nous organiser. C'est pourquoi nous avons décidé de définir des objectifs intermédiaires basés sur le but final de ce projet. Tout cela pour pouvoir à la fois être plus organisé mais aussi de pouvoir définir notre avancement.

Nous avons donc choisi de définir deux objectifs majeurs partitionnés en objectifs mineurs définit ci-dessous :

I) Obtenir une application fonctionnelle sur ordinateur

   a. Implémenter le contrôle par caméra

   b. Transcrire le code en orientée objets

   c. Améliorer l'ergonomie

II) Obtenir une application fonctionnelle sur Raspberry Pi

   a. Définir les spécifications matérielles

   b. Transporter notre application sur Raspberry Pi

   c. Application Android externe pour configurer notre application (optionnel)

B. Organisation

Les objectifs étant posés, nous nous sommes lancés dans la planification du projet. Lors de celle-ci, nous avons réfléchi à comment organiser le déroulement de ce projet dans le but de pouvoir répondre aux objectifs que nous avons indiqués précédemment mais aussi en respectant les contraintes de temps qui nous sont imposées.

Nous avons donc décidé de scinder le projet en différentes phases listées ci-dessous :

- La phase de documentation : lors de cette partie du projet nous nous sommes concentrés sur la rédaction d'un cahier des charges, la recherche d'informations par rapport à différentes technologies ayant un rapport avec le langage Python et le traitement d'images.

- La phase de développement : cette phase représente la plus grande part de notre projet. C'est dans celle-ci que nous avons mis en commun la documentation que nous avions recherché et nos connaissances pour pouvoir développer une application fonctionnelle en répondant aux furs et à mesure à nos objectifs intermédiaires.

- La phase de réalisation d'amélioration : elle représente la fin de notre projet car c'est dans celle-ci que nous listés les différentes améliorations possibles pour l'application.

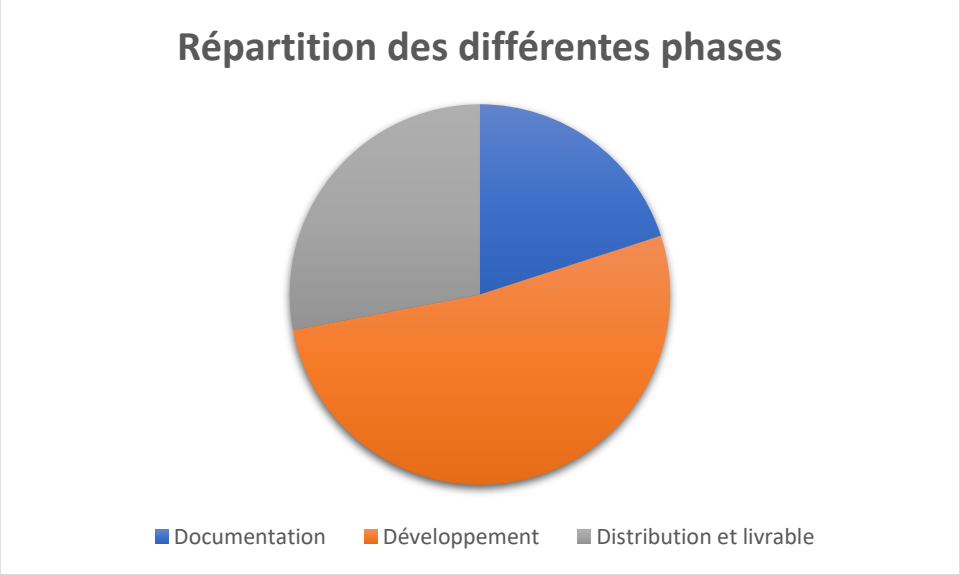


Figure 2:Diagramme circulaire montrant la répartition des différentes tâches

Diagramme de Gantt montrant la planification des différentes phases du projet (cf. Annexe 1).

C. Documentation

Une fois que nous avions déterminé les objectifs que nous allions suivre, nous nous sommes lancés dans une phase de recherche d'informations. Nous nous sommes renseignés sur le langage informatique python et les différentes librairies utilisables pour pouvoir réaliser du traitement d'images ou encore faire du développement de jeux.

Nous avons consulté de nombreux sites de veille technologique concernant le traitement d'images. Notre analyse s'est rapidement portée sur les librairies python Tkinter, PIL ou encore OpenCV.

Nous avons également consulté de nombreux tutoriels sur l'environnement python, comment installer des librairies à partir d'une console Windows ou bien encore comment y accéder depuis un environnement de développement tel que PyCharm.

Cette partie, comme vu dans le diagramme de Gantt a pris un certain temps mais nous a permis de réellement comprendre les mécanismes du langage et des diverses librairies. Cela nous a permis de comprendre en profondeur les subtilités du langage.

2) Développement

Une fois que la documentation concernant les différentes technologies à utiliser était terminée, nous nous sommes attelés au développement du jeu. Pour ce faire, nous avons décidé de commencer par l'implantation du contrôle par caméra permettant de déplacer les raquettes par un mouvement du corps. Ensuite, réétudier le code fourni par M. Fasquel et le modifier intégralement pour l'adapter et ainsi rendre le jeu fonctionnel.

La dernière étape pour avoir un jeu toujours plus fluide a été de reprogrammer le jeu de linéaire à un code orienté objet.

Nous avons décidé après coup d'améliorer l'expérience utilisateur en créant une interface graphique, notamment composé de menu permettant d'adapter l'expérience pour chaque ordinateur et pour différents thèmes.

A. Implémentation du système de caméra

Le code fourni a été programmé en python à l'aide de l'espace de développement PyCharm. Python ne possédant pas de nombreuses librairies par défaut, nous avons installé un interpréteur local contenant diverses librairies qui nous ont été nécessaires.

En effet, une des librairies les plus importantes fut OpenCV, c'est une librairie permettant d'effectuer de nombreuses opérations concernant le traitement d'images. Ainsi après l'installation de cette dernière, nous avons pu commencer la première partie à savoir l'implémentation du système de caméras.

Dans un premier temps, nous avons créé un nouveau fichier pour tester les diverses fonctions sans compromettre le programme à chaque implémentation d'une nouvelle fonction. La première chose effectuée fut de récupérer un flux vidéo à partir d'une webcam, cette étape fut rapide puisque nous avions au préalable effectué quelque chose de similaire lors d'un cours de vision industrielle. Nous avons utilisé la librairie cv2 (OpenCV pour python), à l'intérieur de cette librairie, la classe capture ainsi que la méthode *Read()* ont été utilisées pour accéder à l'image courante d'une caméra.

L'étape suivante consistait à séparer l'image en deux parties distinctes, l'une pour le joueur 1, l'autre pour le joueur 2. Pour ce faire, nous avons utilisé une méthode fournie pour une image capturée à partir d'OpenCV. Plusieurs avantages s'offrent à nous en séparant l'image en deux. Le premier avantage est la facilitation de la mise en place d'une partie de jeu pour les joueurs, notamment par question d'ergonomie car il est plus facile d'avoir un écran chacun.

Le deuxième avantage qui lui a une importance cruciale est la mise en place de deux traitements de données séparées, le joueur 1 et le joueur 2 ne faisant pas les mêmes mouvements, nous avons besoin d'un traitement d'images pour chacun des joueurs. Faire deux écrans distincts est donc un bon moyen de traiter les données séparément.

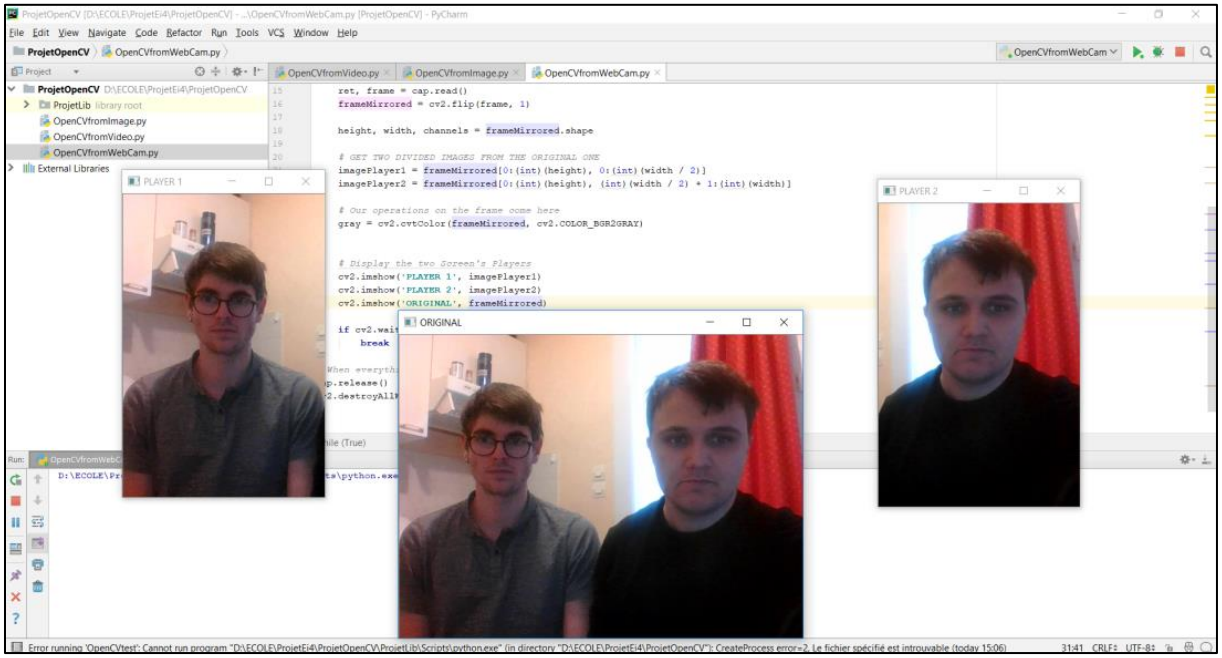


Figure 3: Capture d'écran montrant l'acquisition du flux vidéo d'une webcam et de la séparation de l'image

Une fois l'acquisition d'un flux vidéo par webcam et la séparation de cette image réalisée, nous avons décidé de penser aux diverses fonctions à implémenter pour pouvoir mouvoir les raquettes du jeu Pong. Nous avons donc réalisé un fichier python se nommant Functions.py qui permet de regrouper toutes les fonctions concernant le traitement d'images.

Fonctions assurant le traitement d'images :

- La fonction *compute_barycenter()* permet de calculer le barycentre de l'image 1 et 2 respectivement représentant le mouvement des deux joueurs. Cette fonction prend en entrée l'image courante capturée par la webcam ainsi que la dernière image capturée, un seuil est également en entrée pour pouvoir effectuer un seuillage. Les images sont transformées en nuances de gris dans la fonction pour pouvoir les traiter avec plus de facilité et sont reconverties en BGR (Blue, Green, Red) pour pouvoir les afficher en couleur lors de la partie de jeu.

- La fonction *get_delta_barycenter()* quant à elle est une fonction très simple qui prend en entrée le barycentre de l'image courante ainsi que le barycentre de l'image précédente. Elle retourne tout simplement la différence entre ces deux barycentres. Si la différence de barycentre est positive, le mouvement de la raquette est donc vers le bas. À l'inverse, si la différence de barycentre est négative le mouvement de la raquette est vers le haut.

- La dernière fonction concernant le système de caméras et le traitement d'images est la fonction *display_image()* qui permet juste d'afficher à l'écran la caméra du joueur 1 dans le coin en bas à gauche et celle du joueur 2 en bas à droite par question d'ergonomie.

B. Implémentation du jeu

Alors que la phase d'implémentation du système de caméras touchait à sa fin, nous avons étudié diverses librairies concernant cette fois l'implémentation du jeu en lui-même.

Tkinter est une librairie permettant l'utilisation d'une interface graphique en python, notamment pour la création des raquettes, de la balle ainsi que de l'arrière-plan. Cette librairie est donc indispensable pour pouvoir afficher le jeu à l'écran et posséder une interface graphique simple.

Sans entrer dans les moindres détails, nous allons maintenant étudier le code du programme principal faisant fonctionner l'application. La fonction principale est la fonction update(), c'est une fonction qui est appelée à chaque instant. Elle n'a pas d'arguments en entrée et ne retourne rien. C'est une fonction qui a chaque instant va comparer les barycentres de l'image correspondant au temps « t » et l'image correspondant au temps « t-1 », ceci étant fait grâce à la fonction *compute_barycenter()* et *get_delta_barycenter()* présentées précédemment. Une fois ces fonctions appelées, la valeur correspondant à la différence des barycentres est stockée dans une variable globale « deltaBarycenter » réutilisée plus tard.

Ensuite est vérifiée la collision de la balle contre les deux raquettes ainsi que les parois horizontales de l'écran pour ne pas que la balle sorte de l'écran. Pour ce faire, de simples vérifications de coordonnées sont effectuées pour la position en x et en y de la balle en pixels. Nous devions vérifier la position relative de la raquette droite, de la raquette gauche ainsi que la position verticale de la balle pour qu'il ne sorte pas de l'écran.

Nous faisons ensuite une modification de la trajectoire de la balle si elle est entrée en contact avec un bord ou une raquette.

En parallèle, grâce à la variable globale « deltaBarycenter » nous incrémentons la position Y de la raquette d'un certain pas et selon le signe de cette variable, la raquette va soit monter soit descendre.

Si aucune des conditions de collision n'était vérifiée, alors l'un ou l'autre des joueurs à gagner selon le côté atteint par la balle, ceci étant accompagné d'une incrémentation d'une variable globale « PointJoueur ». Après avoir effectué l'incrémentation des points, nous réinitialisons complétement le jeu grâce à une fonction *reset_game()*.

Cette fonction fait appelle aux fonctions ci-dessous :

- La fonction *game_off()* comme son nom l'indique sert uniquement à affecter une valeur fausse à un booléen global. Ceci permet de mettre le jeu en pause lorsqu'un joueur a marqué un point ou lorsque l'on clique sur la barre espace grâce à un gestionnaire d'évènement.

- La fonction *initiate_game()* sert quant à elle à initialiser le jeu, à savoir la taille de l'écran, le fond d'écran et les couleurs des raquettes et de la balle.

- La fonction *« draw_jeu »* est une fonction comme son nom l'indique qui va créer l'interface graphique du terrain, notamment les rectangles pour les raquettes et la balle ainsi qu'un fond d'écran. Un texte représentant le score des joueurs est également affiché à chaque pause du jeu. Ceci est créé grâce aux méthodes *create_image()*, *create_text()* et *create_rectangle()* de la classe Canvas.

Figure 4: Capture d'écran du jeu lors d'une pause résumant l'appel des diverses fonctions de la classe Canvas

La dernière étape pour économiser des ressources et avoir un jeu plus fluide était la destruction de l'entièreté du Canvas à chaque appel de la méthode *update ()*. Nous avons donc utilisé la méthode *delete(ALL)* de la classe Canvas au début de la fonction *draw_jeu()* pour ainsi détruire le Canvas à chaque instant, ainsi lorsque l'on affiche le Canvas à nouveau, la mémoire a déjà été vidée.

Après avoir implémenté toutes ces fonctions en programmation linéaire, nous avons pu obtenir un jeu fonctionnel et le premier jet était assez satisfaisant. Cependant, la fluidité n'était malheureusement pas toujours au rendez-vous, nous avons donc décidé de passer ce programme en orienté objet pour une meilleure efficacité et un appel aux méthodes plus efficaces.

C. Implémentation d'une version orientée objet

Comme expliqué précédemment, nous recherchions avant tout une fluidité de jeu plus qu'un graphisme de qualité exceptionnelle, c'est pourquoi nous avons commencé l'implémentation d'une version orientée objet.

Le premier sujet de réflexion fut la création d'un diagramme UML représentant les différentes classes du jeu Pong, nous avons gardé quelques parties du code en linéaire par soucis de praticité, ceci nous a donc amené au diagramme UML suivant :

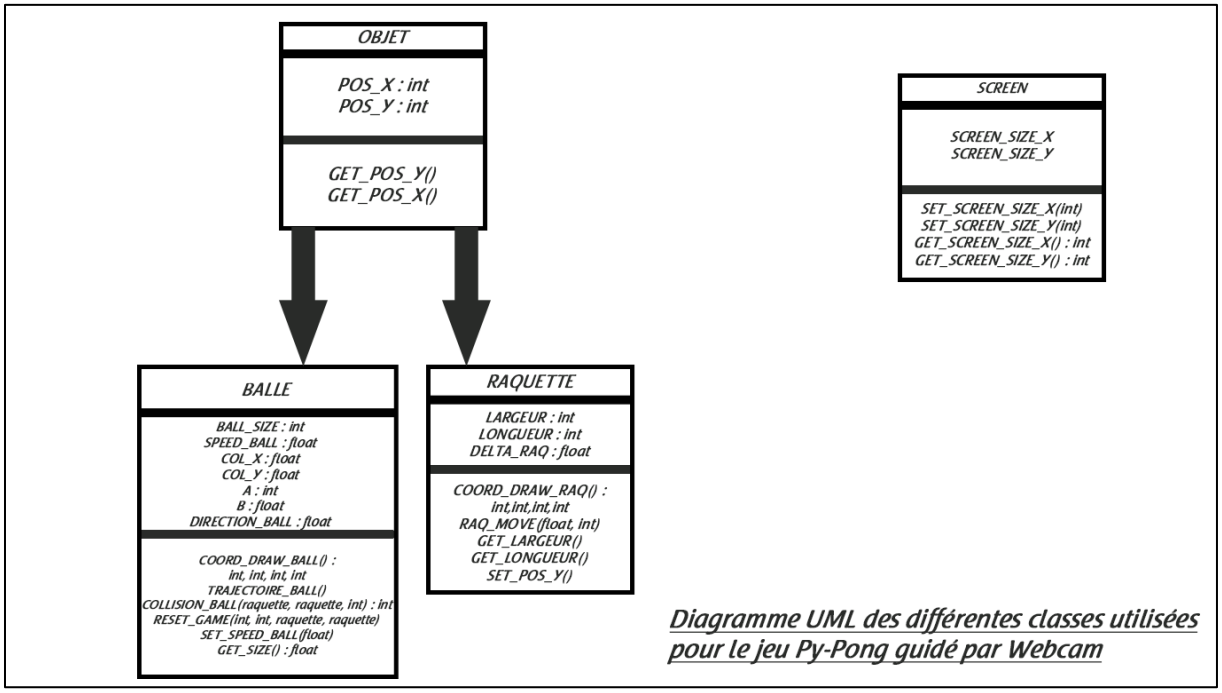


Diagramme UML des différentes classes utilisées pour le jeu Py-Pong guidé par Webcam

Comme on peut le constater, deux attributs et deux getters étaient similaires pour les classes Balle et Raquette, cependant, il n'était pas très intéressant de créer une classe objet pour ensuite la dériver. Nous avons donc tout simplement créé deux classes distinctes « Balle » et « Raquette ». Une classe « Screen » a également été créée représentant tout simplement les dimensions de la fenêtre voulue.

Ci-dessous une explication plus en détails des méthodes composant les classes « Raquette » et « Ball ».

Méthodes de la classe Raquette :

- La *méthode Raq_Move()* prend en entrée la variable globale « delta_barycenter » ainsi que la taille de l'écran. On vérifie à l'intérieur de cette méthode la position en Y pour savoir si la raquette entre ou non en contact avec les bords de l'écran. Si ce n'est pas le cas, on déplace la raquette vers le haut ou le bas en fonction du signe de « delta_barycenter ».
- La méthode *Coord_draw_raq()* quant à elle retourne quatre valeurs, ce sont les valeurs des quatre points du rectangle correspondant à la raquette pour ensuite pouvoir les dessiner avec la fonction *create_rectangle()* de la classe Canvas.
- Les autres méthodes sont simplement des getters et setters pour pouvoir accéder aux attributs de la classe.

Méthodes de la classe Ball :

- La méthode *coord_draw_ball()* comme pour la raquette va retourner quatre valeurs des quatre points du rectangle de la balle pour pouvoir la dessiner.
- La méthode *trajectoire_ball()* sert tout simplement à donner la direction de la balle à chaque instant en régissant plus ou moins une équation « ax + b = y ».
- La méthode *collision_ball()* prend en entrée la raquette gauche et droite ainsi que la taille de l'écran en largeur. En effet, nous allons ici vérifier chaque collision de la balle avec les bords de l'écran ou les raquettes. La méthode retourne donc une valeur spécifique selon quel joueur a marqué le point. La valeur de retour sera mise à jour dans le programme principal et l'incrémentation des compteurs des joueurs également.
- La méthode *reset_game()* régit le jeu en général. Cette dernière prend en entrée la taille de l'écran ainsi que les deux raquettes. Cette méthode sert à réinitialiser la position des deux raquettes ainsi que de la balle. En parallèle, elle initialise la trajectoire de la balle aléatoirement pour le prochain point.

Ces méthodes sont appelées dans le programme principal lors de la fonction *update()* à chaque instant, à l'exception de la méthode *reset_game()* qui est appelée uniquement lorsqu'un joueur a marqué un point.

À la fin de cette implémentation, le jeu étant fonctionnel et plus fluide, le résultat était correct et ne nécessitait pas l'utilisation d'un multi-processing pour pouvoir fonctionner. Nous avons donc gardé cette version et utiliser le temps restant pour améliorer l'expérience utilisateur par le biais d'une interface graphique un peu plus poussée.

D. Implémentation d'une interface graphique

Après avoir terminé la programmation orientée objet, comme expliqué précédemment, nous avons décidé d'améliorer l'expérience utilisateur. Par ceci nous entendons le fait d'ajouter des fonctionnalités permettant de modifier la résolution de la fenêtre du jeu ou de proposer différents arrières plans.

La première idée que nous avons eu était la création d'un menu déroulant pour pouvoir sélectionner la résolution de la fenêtre. Cette solution était très ergonomique pour l'utilisateur et permettait facilement d'avoir accès à tous les choix via le jeu lui-même.

Nous avons donc implémenté des fonctions *set_bool_1080, 720, 480, 360* pour pouvoir changer la résolution, ces dernières supprimant la fenêtre courante et en recréant une nouvelle avec les dimensions choisies. Lors du clic sur le menu déroulant chaque fonction est appelée selon le bouton grâce à une implémentation de menu classique en cascade.

Nous avons fait de même avec les différents arrière-plans proposés dans un deuxième menu tout en adaptant le code pour les adapter aux résolutions. Pour ce faire, nous avons créé quatre images correspondant aux résolutions différentes. Lors de l'appel de la fonction, nous allons chercher l'image qui correspond à la résolution grâce à la méthode *PhotoImage()* de la classe *ImageTk* proposée par la librairie PIL. Cette méthode permet d'acquérir une image à partir d'un chemin spécifié sur l'ordinateur.

Nous avons donc conclu l'implémentation du jeu sur PC par cette amélioration graphique pour l'utilisateur (cf. Annexe 2).

3) Améliorations

A. Fonctionnalités

Une fois que nous avions obtenu une application fonctionnelle et amélioré de façon sommaire l'expérience utilisateurs, nous avons décidé de penser à d'autres fonctionnalités ou améliorations que nous pourrions implémenter pour les prochaines versions du jeu.

Dans les ajouts que nous voulons faire, nous les avons séparés en deux catégories, une relié aux fonctionnements profonds du jeu et une autre qui se centre plus autour de l'aspect visuel.

Fonctionnalités :

- Un ajout qui pourrait être intéressant est la possibilité de donner une impulsion à la balle. C'est-à-dire que le joueur en effectuant une action spécifique comme un cri ou alors un geste en particulier pourrait déclencher une action spéciale. Cette action entrainerait une accélération brutale de la balle à la façon d'un « Air Hockey ».
- Une autre fonctionnalité serait de faire en sorte que la balle gagne en vitesse à chaque fois qu'un joueur réussit à la renvoyer. De cette façon, les parties seraient moins longues et la difficulté serait augmentée.

Partie Graphique :

- Une des améliorations graphiques que l'on voudrait faire serait de pouvoir appliquer des textures sur les raquettes et la balle. De plus, on voudrait pouvoir appliquer des effets de particules derrière la balle, implémenter un son d'ambiance et aussi un son lorsque la balle entre en contact avec les raquettes.

B. Game Engine

Depuis le début du projet, nous pensons qu'il serait intéressant à termes de redévelopper l'application au sein d'un moteur de jeu tel que Unity. C'est pourquoi nous avons listé ci-dessous les avantages et possibles inconvénients que cette transcription pourrait avoir.

Avantages :

- Environnement de développement plus puissant
- Création et gestion des objets plus intuitive (game object)
- Gestion des collision plus précise (collider)
- Création et gestion des menus plus évidentes
- Création d'un exécutable pour Windows, linux, WebGL

Inconvénients :

- Changement de langage de python vers C#
- Existence de librairies comme OpenCV ou tkinter
- Redéfinir les conditions de déplacement et de contact de la balle

Conclusion

Pour conclure, ce projet fut très intéressant et enrichissant car il nous permit de nous améliorer dans de divers domaines. Nous avons pu approfondir nos compétences dans le langage python car bien que nous ayons eu d'autres opportunités de travailler dans celui-ci, ce projet fut un réel challenge. Il nous a donné l'occasion de consolider nos bases tout en découvrant d'autres aspects que nous ne connaissions que très peu. En parallèle de cela, nous avons aussi pu travailler sur une partie traitement d'image, ce qui fut une agréable expérience car cela nous a donnés un avant-gout du potentiel mais aussi de la complexité que ce domaine offre. Le travail d'équipe fut aussi une bonne expérience, nous avons pu voir le bénéfice que cela apporte en matière d'efficacité et de productivité mais aussi des contraintes que cela peut avoir quand les habitudes de travails ne sont pas uniformisées.

L'objectif du projet était de pouvoir développer une version du jeu Pong à la différence que celui-ci serait contrôle non pas clavier ou joystick mais par caméra. À la suite de cela, il fallait pouvoir l'installer sur une plateforme Raspberry Pi. Bien que nous ayons réussi à implémenter la caméra, il nous a été impossible de réussir à l'installer sur une plateforme mobile. L'application offre cependant un résultat fonctionnel même si l'expérience utilisateurs pourrait être améliorée.

Si nous devions recommencer ce projet, il y a de nombreuses choses que nous aimerions changer ou mettre en place lors du déroulement du projet. L'une des premières choses que nous voudrions faire et d'imposer une norme de codage pour que la façon dont on écrit ou tout du moins lie les fonctions entre elles soit normés. Finalement, nous aimerions aussi pouvoir organiser notre projet et étudier au préalable les fonctions que nous allons implémenter pour pouvoir avoir une organisation du code plus claire et structurée.

Bibliographie

Sites web :

OpenCV :

https://docs.opencv.org/3.0beta/doc/py_tutorials/py_gui/py_image_display/py_image_display.html

Tkinter :

http://tkinter.fdex.eu/

PIL :

https://pillow.readthedocs.io/en/5.1.x/

Wikipedia,Pong :

https://fr.wikipedia.org/wiki/Pong

Python :

https://www.python.org/

Conda :

https://conda.io/docs/index.html

Py-Pong guidé par webcam

Projet réalisé par : Alexandre Durandeau et Jérémy Chopin
Projet encadré par : M. Jean-Baptiste Fasquel

Résumé

Super Galactic Ball est un **programme** permettant de jouer à une version modifiée du jeu vidéo **Pong**. Dans ce jeu, les joueurs pourront utiliser une **caméra** au lieu d'un clavier pour pouvoir interagir avec le programme. Cette application fut créée lors d'un projet réalisé par des étudiants en quatrième année à l'ISTIA, école d'ingénieurs de l'université d'Angers, sous la direction de M. Jean-Baptiste Fasquel.

Ce projet avait pour objectif de partir d'un code en **python** du jeu Pong pour pouvoir intégrer un contrôle par caméra. De plus, il fallait pouvoir installer ce programme une fois fini sur un support mobile Raspberry Pi.

Dans le but de pouvoir réussir ce projet, il a fallu faire de nombreuses recherches et tests sur les différentes librairies du langage python. À la suite de cela, l'implémentation du contrôle par caméra fut une réussite et la transcription du code en orientée objet fut nécessaire pour pouvoir avoir un code plus optimisé.

Finalement, l'application est fonctionnelle et l'expérience utilisateurs a été améliorée mais le transport de l'application sur support mobile ne fut pas possible. Des améliorations sont encore possibles et adapter le programme sur un moteur de jeu comme Unity est envisageable.

Mots-clés : programme, Pong, caméra, python

Abstract

Super Galactic Ball a **program** to play a different version of the video game **Pong**. In this version, players will be able to use a camera instead of a keyboard to interact with the program. This application was made by fourth year students from ISTIA, engineering school of the university of Angers, under the lead of Mr. Jean-Baptiste Fasquel.

This project 's aim is to modify a python version of the game Pong to integrate a control by **camera**. Moreover, the program would be finally installed on a Raspberry Pi platform.

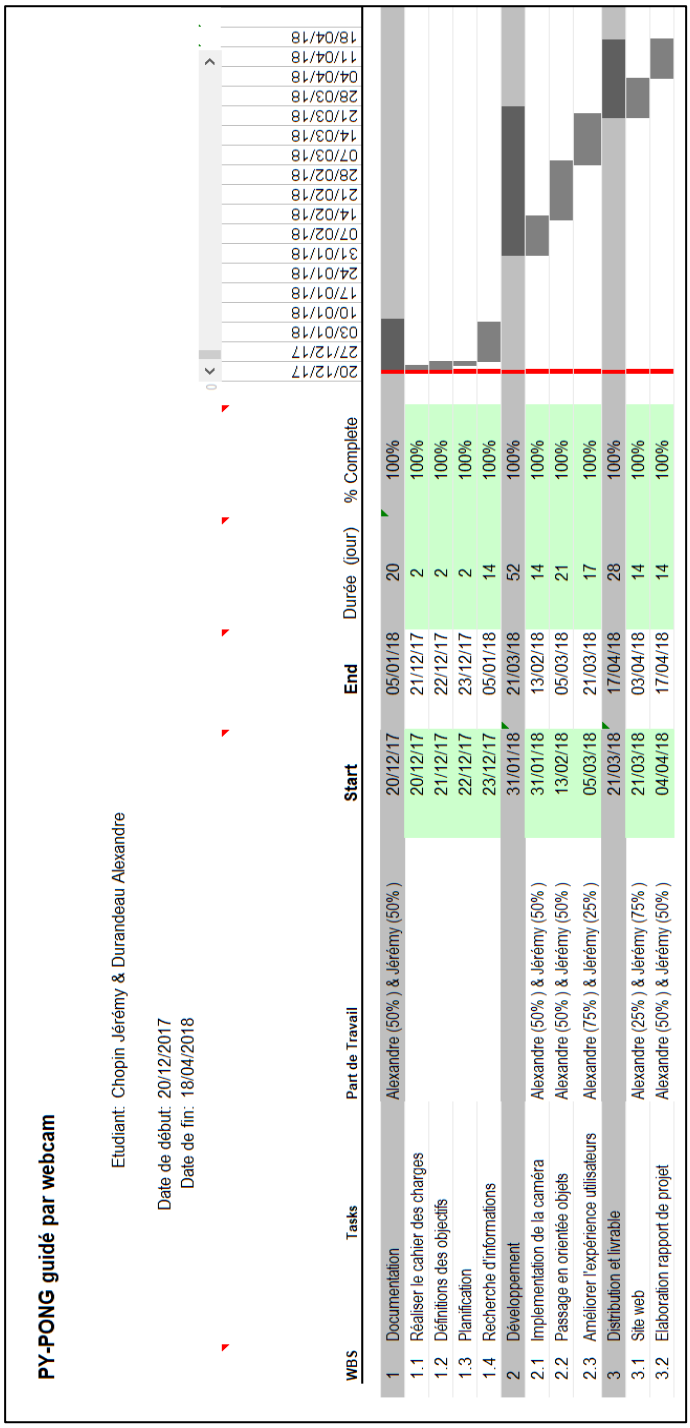
A Lot of research and tests upon **python** language and its libraries were needed. Then the implementation of the control by camera was a success and a transcription of the program into an oriented object program was necessary to be more optimized.

Finally, the application is functional, and the user experience has been improved, but the installation of this one on a mobile platform was not possible. Some upgrades are workable and adapt the program into a game engine as Unity seems feasible.

Keywords : program, Pong, camera, python

Annexe

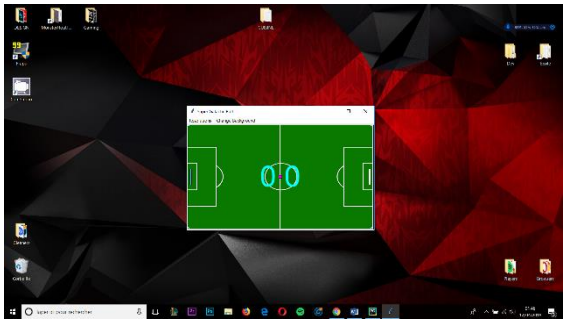
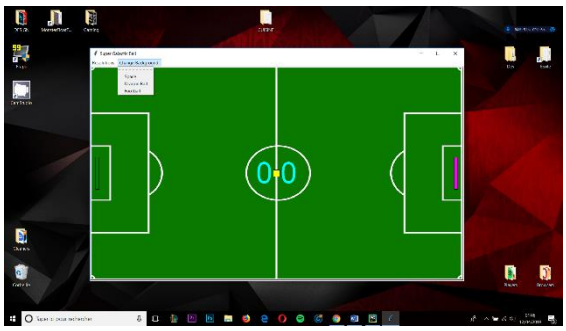
Annexe 1 : Diagramme de Gantt montrant la planification des tâches



WBS	Tasks	Part de Travail	Start	End	Durée (jour)	% Complete
1	Documentation	Alexandre (50%) & Jérémy (50%)	20/12/17	05/01/18	20	100%
1.1	Réaliser le cahier des charges		20/12/17	21/12/17	2	100%
1.2	Définitions des objectifs		21/12/17	22/12/17	2	100%
1.3	Planification		22/12/17	23/12/17	2	100%
1.4	Recherche d'informations		23/12/17	05/01/18	14	100%
2	Développement		31/01/18	21/03/18	52	100%
2.1	Implémentation de la caméra	Alexandre (50%) & Jérémy (50%)	31/01/18	13/02/18	14	100%
2.2	Passage en orientée objets	Alexandre (50%) & Jérémy (50%)	13/02/18	05/03/18	21	100%
2.3	Améliorer l'expérience utilisateurs	Alexandre (75%) & Jérémy (25%)	05/03/18	21/03/18	17	100%
3	Distribution et livrable		21/03/18	17/04/18	28	100%
3.1	Site web	Alexandre (25%) & Jérémy (75%)	21/03/18	03/04/18	14	100%
3.2	Elaboration rapport de projet	Alexandre (50%) & Jérémy (50%)	04/04/18	17/04/18	14	100%

Annexe 2 : Image du jeu avec différents arrière-plan et résolution

Arrière plan : Terrain de football

Arrière plan : Espace

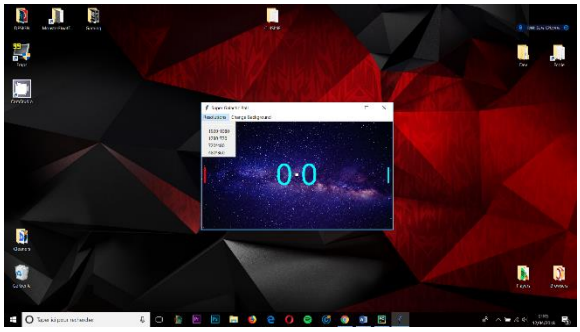
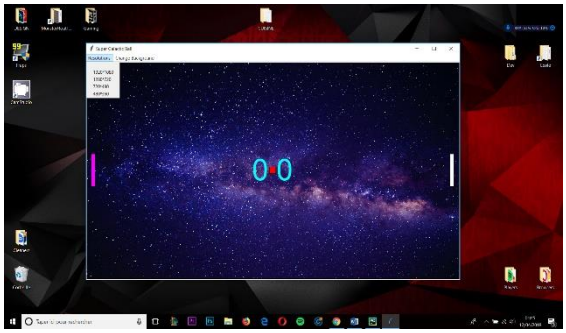



*Figure 5: Ci-dessus, le jeu composé de différents arrière-plans en résolution 853*480 à gauche et 1280*720 à droite*