420-V31-SF
PROGRAMMATION DE JEUX VIDÉO III
TRAVAIL PRATIQUE #2
FURY
PONDÉRATION : 15%

OBJECTIFS PÉDAGOGIQUES

Ce travail vise à familiariser l'étudiante ou l'étudiant avec les objectifs suivants :

- Appliquer une approche de développement par objets (compétences 016T)
- Apporter des améliorations fonctionnelles à une application (compétence 0176)
- Concevoir et développer une application dans un environnement graphique (compétence 017C)

De même, il permet à l'étudiante ou à l'étudiant d'appliquer les standards de conception, de documentation et de programmation.

MISE EN CONTEXTE ET MANDAT

Ce travail pratique a pour but premier de créer un jeu complet en 2D (Affichage sprite). L'étudiant devra donc, suite aux spécifications de l'application, définir une architecture orientée objet, où l'héritage et le polymorphisme sont prépondérants. Il pourra par la suite utiliser des éléments de code fournis afin d'accélérer le développement de l'application.

Par la suite, dans un environnement C++ / SFML, l'étudiant devra développer un jeu complet et en assurer la qualité à la fois par des tests d'utilisation et des assertions.

SPÉCIFICATIONS DE L'APPLICATION DE BASE

Vous allez créer une sorte de successeur d'un jeu classique sur borne d'arcade et sur Atari 2600 : Berserk, et de son successeur moins connu : Frenzy. À L'origine ces deux jeux étaient des jeux illimités où il fallait marquer le plus grand nombre de points possible. Nous allons en faire quelque de plus moderne (mais à peine).

Ce sera une bonne occasion de mettre en pratique les concepts d'héritage et de polymorphisme, d'utiliser les premiers design patterns que nous avons exploré en classe et, éventuellement, d'utiliser des animations de sprite (bien que ce dernier point ne soit vraiment pas prioritaire)

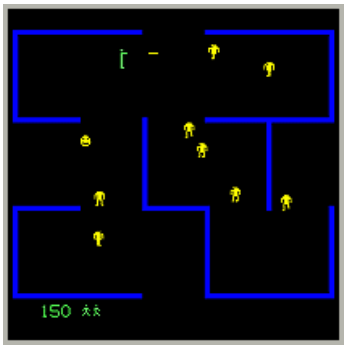
Nous allons privilégier le mode de développement agile. À la fin de chaque sprint, le jeu devrait être dans un état fonctionnel et jouable. Pour chaque sprint, différentes user-stories seront soumises, et des conseils de développement seront aussi suggérés.

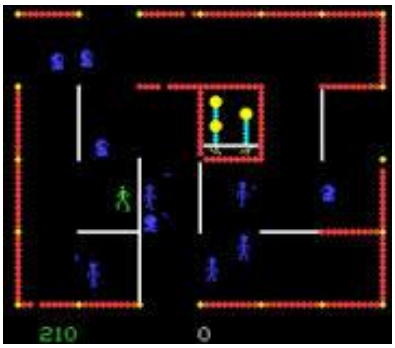
INTERMÈDE : UN PEU D'HISTOIRE ET DES ANECDOTES.

Il y a très, très longtemps, à une époque très lointaine, pour jouer à des jeux vidéo, on utilisait ceci :

Ou alors, si vous vouliez un peu plus de qualité, et si vous en aviez les moyens, vous pouviez aussi utiliser ceci :

À cette époque préhistorique, des jeux comme Berserk, et son successeur, Frenzy, étaient parmi les jeux les plus joués qui soient.

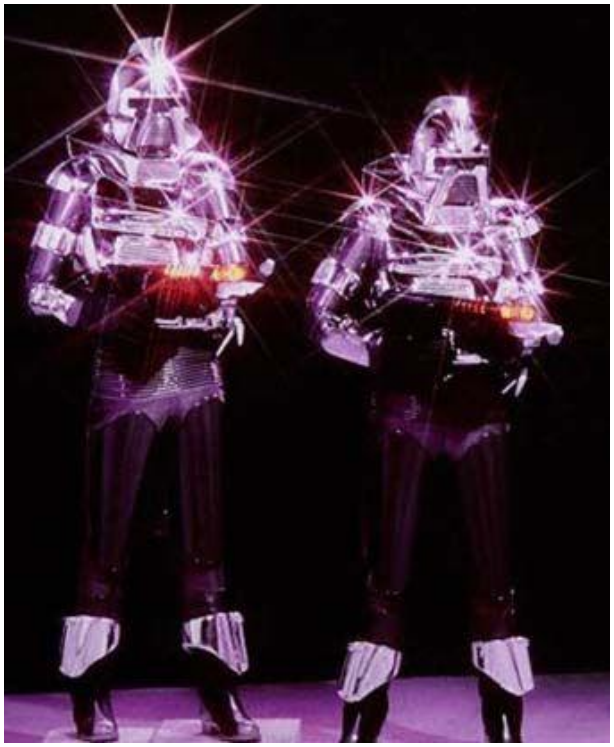




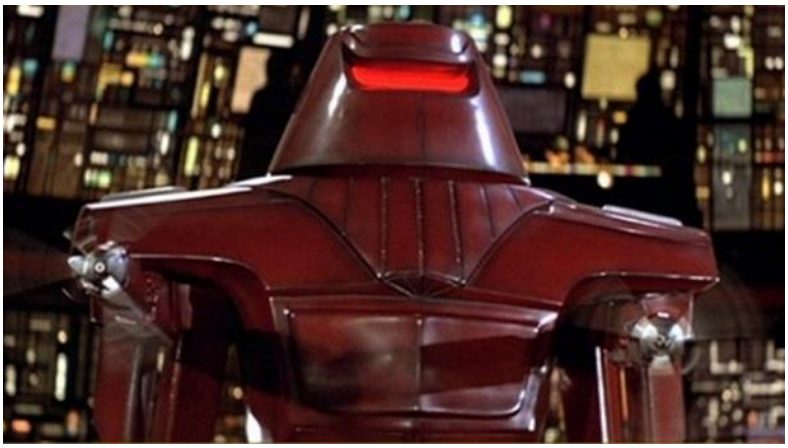
Alors ça fera un joli baume sur le cœur de pépé du jeu vidéo de votre gentil coach, qui a joué à ces jeux comme petit gamin (en fait juste à Berserk sur Atari 2600, mais ne gâchons pas l'effet dramatique…) de vous voir créer le troisième opus de cette glorieuse série, j'ai nommé FURY (parce que le titre, il va bien avec les deux autres noms, ne trouvez-vous pas?)

Pour le look des robots de Berserk, ils sont dans la lignée de ce qui fut populaire à l'époque : les cylons : les robots méchants de la première mouture de Battlestar Galactica (https://www.youtube.com/watch?v=oYZcZSVYx3M), ou encore Maximilian, le robot psychopathe de l'excellent et méconnu film de Disney : Le trou noir (https://www.youtube.com/watch?v=pi_ZA1JHGNE) …qui n'a strictement rien d'un film de Disney. Sans joke, pour un kid de 6 ans ce film était cauchemardesque. L'intro du film donne le ton : https://www.youtube.com/watch?v=IJaypC51Dds

Les Cylons



Maximillian



Bon, revenons au cours normal de ce TP…

DESCRIPTION DU JEU D'ORIGINE - BERSERK

Berserk est donc un jeu ou on parcourt un labyrinthe. La disposition des murs de chaque pièce est générée au hasard. On doit détruire les robots de chaque pièce afin de marquer le plus grand nombre de points possible. À tout moment, je joueur peut sortir de la pièce par l'une des portes, sauf celle d'où il arrive qui est fermée. Si le joueur reste trop longtemps dans la pièce, un fantôme indestructible en forme de smiley souriant, "Evil Otto" entre en jeu et se dirige vers le joueur en bondissant joyeusement.

Les murs sont électrifiés. Si le joueur ou un robot, entre en contact avec un mur, il meurt (bien qu'à la base le robot essaye d'éviter les collisions avec un mur, il peut lui arriver de le faire). Si un robot entre en contact avec un autre robot ou le joueur, les deux meurent. Un robot qui tire sur un autre robot le tue. Le joueur en collision avec l'explosion d'un robot… bref vous avez compris le concept toute collision provoque la mort de tout le monde qui est impliqué! Radical? Violent? Disons une solution simple pour gérer les collisions ou la mémoire et les capacités machines étaient ridicules comparées à aujourd'hui.

À noter que le nombre de projectiles ennemis est partagé entre tous les robots. On ouvre à zéro projectile dans la première salle et à mesure que le jeu avance les robots ont droit à deux, trois, quatre, puis cinq projectiles (pour tous les robots). Qui tire les projectiles? L'algorithme est très simple :

```
À chaque cycle d'affichage
{
  Pour chaque robot
    [Si le joueur est dans un axe de tir du robot] ET [S'il s'est
    écoulé plus d'une demi-seconde depuis le dernier tir de ce robot
    précis]
      On ajoute le robot à la collection de tireurs

  On brasse la collection de tireurs

  Pour chaque tireur jusqu'à la limite de projectiles disponible
     le tireur fait feu
}
```

Accessoirement ça donne l'effet comique que si un autre robot était entre le joueur et le tireur, les robots ennemis ont la sale tendance de se tirer entre eux… un joueur habile saura utiliser cette situation.

Les déplacements étaient fait sensiblement de la même manière : toujours vers le joueur, à la verticale, à l'horizontale ou en diagonale, selon l'axe le plus rapide, bien que les robots, on le rappel, font des efforts de base pour éviter les murs.

Tous comme les tirs, la vitesse de déplacement est aussi répartie entre les robots. Moins il y a de robots, plus ils sont rapides.

Petite anecdote intéressante, numériser des phrases à l'époque de Berserk coûtait… 1000$ du mot (en procédure, logiciels, support mémoire, etc.) les concepteurs ne sont tout de même lancés…

Pour essayer Berserk (version arcade) : *https://archive.org/details/arcade_berzerk*

Pour essayer Berserk (version Atari 2600) : *http://www.arcadeboss.com/game-large-24-7-Berzerk.html*

Référence : https://en.wikipedia.org/wiki/Berzerk_(video_game)

Référence : *http://www.hardcoregaming101.net/berzerkfrenzy/berzerkfrenzy.htm*

DESCRIPTION DU JEU D'ORIGINE - FRENZY

Frenzy fut moins populaire, entre autre parce qu'il était considéré comme trop difficile (le Dark Souls de son temps en quelque sorte…) ici les murs électriques sont évacués et on ajoute les murs destructibles et miroirs. Les collisions ne provoquent plus la mort de personne (sauf avec les explosions de robot) et on ajoute quelques petits "boss fights" qui se produisent environ toutes les quatre pièces.

C'est le grand nombre de robots à l'écran et les murs destructibles, faisant en sorte que le joueur n'était plus vraiment à l'abri nulle part qui rendit ce jeu si difficile.

À noter qu'ici, "Evil Otto" était destructible. Il avait besoin de trois tirs pour être éliminé. Les deux premiers le ralentissaient, et le troisième le terminait, mais il revenait immédiatement, et plus rapide à chaque fois.

Pour essayer Frenzy (version arcade) : *https://archive.org/details/arcade_frenzy*

Référence : *https://en.wikipedia.org/wiki/Frenzy_(video_game)*

AVANT DE COMMENCER – ESSAYER LES JEUX DE BASE

Vous devez prendre du temps pour essayer les jeux de base et vous familiariser avec le sujet. Il s'agit d'un travail sérieux. Tout développeur d'un jeu qui se base sur un autre se doit d'être familier avec le sujet d'origine. Ce n'est pas quelque chose à prendre à la légère.

AVANT DE COMMENCER – CHOIX DES CONTRÔLES ET DE LA RÉSOLUTION

Clavier ou Gamepad? Le choix est vôtre. Idéalement les deux, mais ce n'est pas du tout exigé. Prenez une résolution maximale autour de 1280 x 720 (720p) afin de maximiser la compatibilité de votre jeu. La souris est seulement nécessaire si on implémente un élément optionnel du sprint 5.

DÉVELOPPEMENT DU JEU FURY (BERSERK III)

SPRINT 1 – USER STORIES

1- Le joueur voit une pièce, celle-ci contient des murs, ainsi que quatre sorties sur les quatre murs de la pièce.

2- Le joueur voit son personnage, près de la sortie gauche. Le personnage peut se déplacer dans la pièce. Il peut se déplacer dans les "8 directions classiques d'un gamepad." S'il entre en collision avec un mur, il est arrêté il ne peut pas progresser dans cette direction.

3- Si le joueur sort de la pièce, l'écran passe au noir pendant une seconde environ (rien n'est affiché) et une nouvelle pièce est créée, avec des murs disposés différemment. Le joueur est positionné à l'opposé de la sortie qu'il a prise.

4 -Si le joueur appui sur le bouton de tir, un tir est provoqué dans l'une des 8 directions (la dernière appuyée si le joueur n'appui sur rien à ce moment-là). Le joueur doit attendre un quart de seconde avant de pouvoir tirer à nouveau. Au départ, deux projectiles du joueur peuvent être à l'écran en même temps. Un projectile est éliminé s'il entre en collision avec quelque chose.

5- Le joueur voit 2 à 12 robots dans la pièce (1 à 6 au hasard + 1 à 6 au hasard).

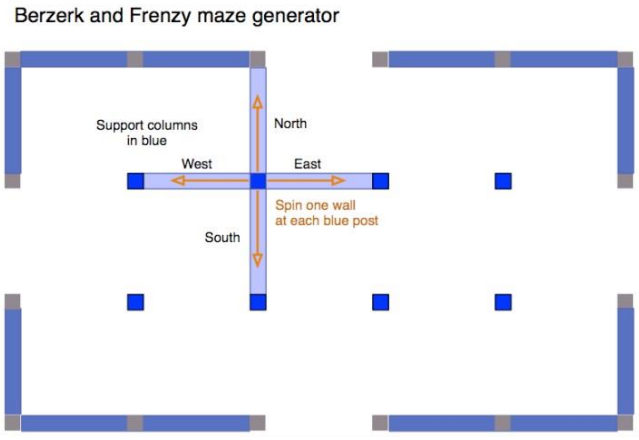
6- Les robots se rapprochent du joueur de manière directe, à la verticale, à l'horizontale, ou en diagonale. S'il y a collision potentielle, ce chemin est éliminé. Si aucun de ces chemins n'est possible, le robot ne bouge pas.

7- Si un robot est atteint par un projectile, le robot est éliminé.

<u>Contraintes de programmation</u>

Robot et Joueur/Humanoïde devraient hériter de la même classe : Personnage, qui serait abstraite (une méthode de déplacement serait une bonne méthode abstraite).

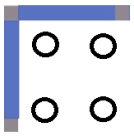
Voici comment les murs sont générés dans le jeu. Chaque mur devrait faire 220 pixels de long par 15 pixels de large. Les carrés pivot seront des petits murs de 15x15 Ce qui fait un labyrinthe de 1190x720 pixels, ce qui laisse une colonne de 90 pixels de large par 720 haut pour les diverses infos (score, vies, bombes, bonus récoltés, etc.).

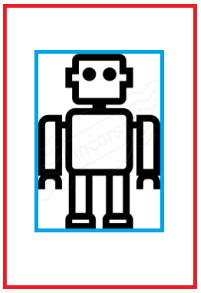


Outre les murs obligatoires autour de la pièce, pour chaque pivot, le labyrinthe

place un mur dans une des quatre directions. Si un mur est déjà présent dans cette direction, il en choisi une autre. (Référence : http://www.robotron2084guidebook.com/home/games/berzerk/mazegenerator/)

Les robots apparaissent dans chaque segment de pièce un peu de cette manière, en bas à gauche (schéma des spawning points par segment). Vous n'êtes pas obligé d'en placer autant, juste assez pour avoir un potentiel maximal de 12 robots. Mais en mettre plus que 12 favorise les positions diversifiées pour les robots.





Pour les robots, créez une "warning box" (intRect) qui est un peu plus grande que la taille du robot. Si la warning box collisionne avec un mur ou un autre robot, ce dernier refusera de considérer cette direction (améliore les collisions et évite de se retrouver avec des "tapons de robot") et elle aura d'autres utilités par la suite. Attention : ce rectangle ne se déplacera pas par magie. Il devra se déplacer avec le robot par programmation.

Vous pouvez "hardcoder" les 8 axes de tir et de déplacement (0, 0.25Pi, 0.5Pi, 0.75Pi, Pi, 1.25Pi, 1.5Pi, 1.75Pi).

Nouveaux assets requis pour le sprint 1 : Personnage, robot, projectile, mur vertical, mur horizontal, mini-mur. **Chaque texture ne devrait être chargée qu'une seule fois.**

Note importante sur les assets : À ce stade-ci et jusqu'à nouvel ordre, sur la beauté des assets et le fait qu'ils soient animés ou non, tout ceci n'a strictement aucune importance. À la limite, vous pourriez travailler même avec des rectangles de couleur et ça fonctionnerait (bon on exagère, mais à peine). L'important est qu'on distingue bien quoi est quoi.

SPRINT 2 – USER STORIES

1- Les robots peuvent tirer sur le joueur. Si un robot est dans l'axe du joueur (peu importe quel axe), il cesse de bouger et devient un tireur potentiel. 0 à 5 tirs sont partagés entre eux. Combien sont possibles?

(Nombre de salles déjà parcourues / 3), round up, maximum de 5 tirs (on considère dans le jeu, que le "hive mind" des robots s'améliore à mesure qu'il s'adapte à affronter le joueur)

Le joueur constate que les robots peuvent tirer, peu importe qu'il y ai des murs ou d'autres robots entre eux et le joueur.

2- Si le joueur est atteint par un projectile. Il perd une vie. Il commence avec 3 vies.

3- Si le joueur perd une vie, fait une transition comme par changement de pièce (une seconde sans rien afficher), puis on recommence dans la pièce actuelle, par la même entrée, et on voit que de nouveaux robots sont recrées(nombre et positions)

4- Quand le niveau commence, il y a une seconde de délai avant que les robots "commencent à jouer."

5- Les robots partagent de la vitesse. Plus il y a de robots, plus ils sont lents.

6- Un robot détruit fait apparaitre une explosion pendant une seconde. Entrer en collision avec une explosion fait perdre une vie. Les robots ne respectent pas les explosions pour ce qui est des collisions et pourraient entrer dedans.

7- Un mur apparait derrière le joueur et bouche l'entrée par laquelle il est arrivé

8- Le joueur évolue dans un labyrinthe de salles de taille X par Y. Sa position de départ est déterminée au hasard. Le joueur n'a aucune idée de la taille du labyrinthe, mais s'il revient dans une pièce qu'il a déjà explorée, elle garde la même disposition, mais les robots sont recrée à neuf (nombre et positions). Si une pièce se trouve à une limite du labyrinthe (haut, bas, gauche, droite), la sortie dans la dite direction est automatiquement fermée.

9- Le joueur voit qu'il peut marquer des points. 50 points par robot abattu, plus 10 points par robot s'il élimine tous les robots d'une pièce. Pour chaque 5000 points, le joueur gagne une vie. Les points sont affichés quelque part. Le joueur reçoit les points peu importe qui détruit les robots.

<u>Contraintes de programmation</u>

L'algorithme de tir des robots est présenté en page 2. Calculez l'angle du robot avec le joueur par tangente inverse (voir corrigé exercice Pokémon2) et déterminez un seuil de tolérance entre l'angle effectif et l'axe idéal (essayez avec 0.18Pi pour commencer, et ajustez au besoin)

Vous devriez vous coder des cheats (raccourci clavier) pour fixer facilement le nombre de tirs des robots.

Le fait d'arriver à zéro vie n'a pour le moment aucune conséquence.

La taille X et Y du labyrinthe peut-être constante, 6 x6 est un bon départ, mais vous pouvez la réduire pour faire des tests.

Chaque pièce doit être créée la première fois que le joueur entre dedans. S'il revient ce sera la même pièce. Considérez-donc un tableau de pointeurs de pièces.

Dans la version finale, le joueur ne saura jamais la position de la pièce dans le tableau / labyrinthe. Mais en développement, vous pouvez l'afficher quelque part.

Nouveaux assets requis pour le sprint : Explosion

SPRINT 3 – USER STORIES

1- Il existe plusieurs types de robots.

- Shooter (niveau 1): le robot par défaut que l'on avait jusqu'ici.

- Brawler (niveau 1): plus rapide que le shooter, ne tire pas (donc ne s'arrête jamais pour ça), mais une collision avec lui est considérée comme un coup. Va respecter les autres robots et les murs (ne s'avance pas si ces derniers sont dans sa warning box).

-Sniper (niveau 2): sous-catégorie du shooter. Ses projectiles ignorent les murs et les autres robots.

-Gunner (niveau 3): sous-catégorie du shooter, tire trois projectiles en parallèle, mais seul celui du milieu compte dans le total des projectiles tirés. Il faut deux tirs pour l'abattre.

- Shieldbot (niveau 2): sous-catégorie du brawler. Se comporte comme lui mais diminue la force des projectiles qui passe près de lui. Si un projectile passe à 100 pixels de rayon du centre du robot, sa force est réduite de moitié. Un projectile de ce genre porte donc l'équivalent d'un demi-coup. Ça vaut pour tous les projectiles.

-Juggernaut (niveau 3): sous-catégorie du brawler. Ne respecte pas les autres robots et les murs et peut entrer en collision avec eux (donc détruit aisément les murs destructibles et les autres robots). Il est immunisé aux murs électriques et aux coups des autres brawlers. Il faut quatre coups pour l'abattre.

2- Les robots choisis le sont au hasard. Au début, on a que des robots de niveau 1, mais plus le score du joueur n'est élevé et plus les robots de haut niveau apparaissent. Le score donné par un robot est multiplié par son niveau.

3- Les robots brawlers et dérivés sont plus rapides que les shooters, mais tous les robots sont plus rapides s'il y a moins de robot. La vitesse de base d'un brawler et de ses sous-classes est juste multipliée par un facteur.

4- Il existe plusieurs types de murs.

- Le mur de base, qu'on avait jusqu'à maintenant.

- Mur électrique : un contact avec un mur électrique provoque la mort.

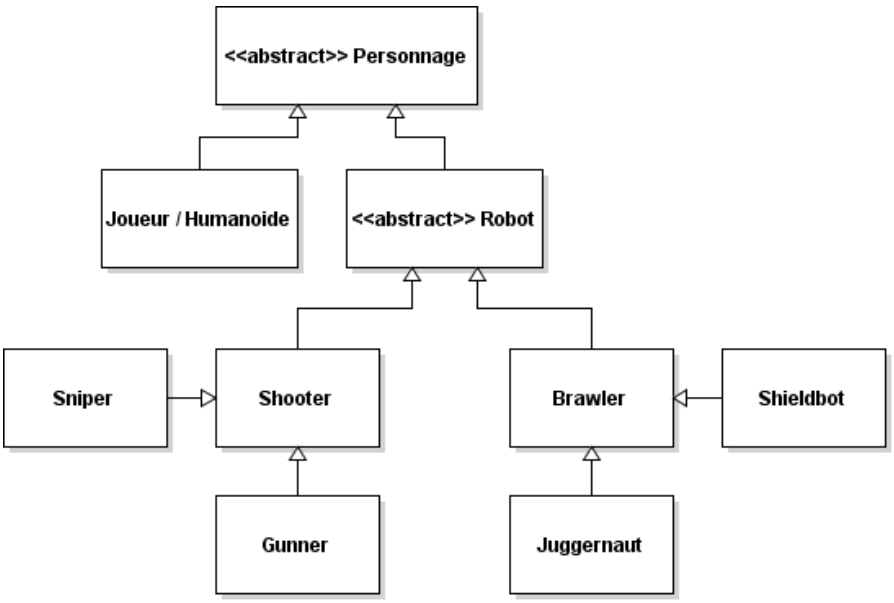
-Mur miroir : Tirer sur un mur miroir reflète le tir. À partir de maintenant, les projectiles ont une durée de vie de 4 secondes, pour éviter les réflexions perpétuelles entre deux miroirs parallèles.

-Mur destructible : composé en fait de plusieurs murs de petite taille : chacun de ces murs peut être détruit par un coup, peu importe la nature.

5- Les murs sont créés au hasard, mais un mur extérieur ne peut pas être un mur destructible.

Contraintes de programmation

Votre diagramme de classe entre les personnages à ce stade-ci devrait ressembler à ça (et personnage peut hériter de sprite si vous le souhaitez) :



À ce stade-ci les robots devraient être crées à l'aide d'une fabrique, standard ou statique. Dans le cas d'une fabrique statique, le score pourrait être un paramètre qui remplacerait le ticket et c'est la fabrique qui décidera si on tend vers les robots de base ou les robots avancés.

Voici une formule proposée : on tire un nombre au hasard entre 0 et (99 + (Score / 50, max 200)) – Vous pouvez en choisir une autre selon votre convenance.

Si entre 0 et 99 : robot niveau 1 / Entre 100 et 199 niveau 2 / Entre 0 et 299 : Niveau 3.

Les trois murs spéciaux devraient hériter du mur de base. Pensez à une fabrique de murs intérieurs et une autre de murs extérieurs (qui ne comprendrait pas le mur destructible). En cas de fabrique statique, ne pas mettre de ticket et laisser le hasard choisir le mur.

Dans tous les cas, si vous voulez charger une texture différente par classe de personnage, pensez à mettre la texture privée et non protected. Par contre la méthode pour charger la texture peut être publique et virtuelle (devra être réécrite pour chaque robot vue qu'elle ne manipule pas la même texture, même si le code risque d'être identique)

Nouveaux assets requis pour le sprint : Tous les nouveaux types de murs et de robots, quoi qu'à part le mur destructible composé de petits murs, vous pouvez tout simplement changer le filtre de couleur de la texture. Voir ce tutoriel pour savoir comment : http://www.sfml-dev.org/tutorials/2.0/graphics-sprite-fr.php

SPRINT 4 – USER STORIES

1- Il existe une porte de sortie. Elle est placée au hasard dans le labyrinthe et se trouve au milieu de la pièce. Si on la prend, on termine le jeu. Cette pièce a une configuration spéciale qui n'est pas générée au hasard. À vous d'imaginer sa disposition précise et le nombre de robots qu'elle contient.

2-Evil Otto arrive dans le jeu. Il s'agit d'un fantôme qui apparait à la porte du joueur après 30 secondes, se dirige lentement vers le joueur (pas obligé de le faire bondir pour rien, seulement si vous le voulez). Il détruit immédiatement tout robot avec lequel il fait contact. Il est plus rapide s'il n'y a plus de robots. Peut recevoir trois tirs avant de mourir. Les deux premiers le ralentissent, et le troisième l'abat. Il revient à nouveau après temps / 2, à vitesse augmentée. Il revient à ses valeurs de départ dans une nouvelle pièce. Il n'est jamais présent dans la pièce de sortie finale.

3- Si le nombre de vies du joueur tombe à zéro, il a perdu et la partie est terminée

Contraintes de programmation

La pièce spéciale devrait être une sous-classe d'une pièce

Si vous sortez par la sortie finale, au minimum, affichez au moins le mot "victoire" quelque part. Si vous perdez votre dernière vie, vous ne respawez plus et inscrivez au moins "partie terminée" quelque part.

Pour déterminer où se trouve la porte de sortie, tirez trois positions de pièce au hasard et choisissez au final la plus éloignée du joueur.

Evil Otto doit être un singleton.

Nouveaux assets requis pour le sprint : Evil Otto,

SPRINT 5 – USER STORIES

1- Nouvelle manière de se comporter pour le shieldbot. Se comporte comme un brawler mais invoque un bouclier sphérique dès qu'un projectile s'approche de lui. Ne peut pas bouger tant que le bouclier est actif. Le bouclier absorber un nombre limité de coups avant de tomber, mais le bouclier est beaucoup plus grand que lui (peut potentiellement protéger d'autres robots derrière lui). Le bouclier s'active dès qu'un projectile s'approche, sans distinction de qui le tire (le bouclier arrête les balles de sniper). Ne divise plus la force des projectiles par 2.

2- Quatre bonus sont placés au hasard dans le labyrinthe que le joueur peut ramasser. Un bonus dans une pièce se trouve à l'endroit inverse où entre le joueur, peu importe la direction où il entre (peut changer d'endroit s'il revient dans la même pièce par une autre porte alors qu'il n'avait pas encore ramassé le bonus). Ces pièces ont toujours les 12 robots maximum.

- Deux des bonus permettent d'ajouter un tir (3 projectiles à l'écran, puis 4).

-Un bonus permet au joueur d'utiliser un bouclier, comme le nouveau shieldbot. Le joueur ne peut pas bouger quand il active le bouclier.

-Un bonus ajoute un "targeter", contrôlé par la souris ou le joystick droit du gamepad. Le tir du joueur n'est plus droit devant, mais en direction du targeter.

3-Le joueur a des bombes. Il en a une au départ et peut en accumuler jusqu'à 3. Un robot détruit a 5% de chance par niveau de laisser tomber une recharge de bombe. La recharge disparait après 10 secondes.

4- Si un joueur dépose une bombe, elle met 5 secondes avant d'exploser, mais explose immédiatement si un robot entre en collision avec elle ou si elle reçoit un projectile. La bombe a 150 pixels de rayon et porte un coup à tout ce qui est dans ce rayon, qu'il soit derrière un mur ou non.

<u>Contraintes de programmation</u>

Les bombes détruisent les murs destructibles, mais ne respectent pas les autres murs par souci de simplicité.

Le bouclier du shieldbot s'active dès qu'un projectile entre dans sa warning box. Celle-ci devrait donc être un peu plus grande que celle d'un autre robot. Pour illustrer le bouclier, vous pouvez utiliser un circle de SFML. Le bouclier a 1800 points et se recharge d'un point par tick d'écran et se décharge aussi de 1 point par tick d'écran quand il est utilisé. Il perd 600 points quand il se prend un coup.

<u>Nouveaux assets requis pour le sprint :</u> Marqueurs de bonus, marqueurs de bombes, bombes, grande explosion, targeter.

SPRINT 6 (OPTIONNEL) – USER STORIES

1- Dans les salles où l'on trouve des bonus pour le joueur se trouvent des usines à robot. Chaque usine prend environ 20 coups avant de mourir et crée un robot aux 3 secondes environ.

2-La porte finale est gardée par un boss. À vous de déterminer la nature du boss. Il devrait hériter lui aussi de robot, mais devrait être énorme, devoir être abattu après plusieurs coups.

3- En commençant le jeu, on a un écran de départ digne de ce nom qui nous offre de commencer le jeu, aller à la fenêtre des options ou sortir du jeu.

4- la fenêtre des options nous offrent un jeu facile (carte 6x6, projectiles des robots lents), moyen (carte 8x8, projectiles des robots à vitesse moyenne) et difficile (10x10 projectiles rapides). On peut aussi retourner à la page de titre.

5- Si on gagne le jeu, on a droit à un bel écran de victoire, puis, on retourne à l'écran de départ. Si on meurt, on a un écran "Game Over" et on retourne aussi à l'écran de départ.

Contraintes de programmation : Aucune

Nouveaux assets requis pour le sprint : Usines, Boss, Écrans de titre, d'option et de sortie.

SPRINT 7 (OPTIONNEL) – USER STORIES

1- Amélioration des graphismes et utilisation des animations.

2-Améliorations des transitions d'écran.

3-Ajout d'éléments sonores.

4- Gamepad / souris et gamepad supporté et choix des commandes supporté aussi.

Contraintes de programmation : Aucune

Nouveaux assets requis pour le sprint : Au besoin.

SPRINT 8 (TRÈS OPTIONNEL) – USER STORIES

1- Les bombes n'attaquent plus ce qui se trouve derrière les murs.
2- Robots plus habiles à trouver le joueur (spécialement les brawlers et dérivés)
3- Sauvegarde des meilleurs scores.
4- Sauvegarde et rechargement de la partie.

Contraintes de programmation : Aucune

Nouveaux assets requis pour le sprint : Au besoin.

CONTRAINTES GÉNÉRALES AU NIVEAU DE LA PROGRAMMATION

- Respectez les standards de programmation dans la nomenclature et dans l'encapsulation.
- Faites des fonctions courtes et concises.
- Utilisation correcte du C++. Exploitation judicieuse des possibilités du langage (pointeurs et références, fichiers .h et .cpp)
- Documentez votre code
- Utilisez les constantes autant que possible (grosse lacune dans le TP1 et les exercices)
- Utilisez les assertions aussi souvent que nécessaire (pré-conditions et post-conditions)
- Utilisez les méthodes et attributs constantes autant que possible (je serai sévère cette fois-ci)
- Chargez le moins de textures possibles (pensez à les mettre static)

ASSETS GRAPHIQUES

Cette fois-ci c'est à vous de trouver et d'implémenter vos assets. Un personnage de base et un robot de base vous sont fournis. Vous pouvez les utiliser au besoin, mais sachez que le rectangle de texture est plus grand que l'image elle-même. De créer un second rectangle par-dessus, porté par l'objet, pour la collision serait de bon ton. Dans ce cas, le rectangle de la texture peut servir de warning box.

BONUS

L'essentiel du TP est dans les sprints 1 à 4. Les sprints 5 à 8 sont divers éléments de bonus. Ou carrément ajoutées au projet dans certaines situations exceptionnelles.

JOURNAL DE PROJET

Vous devez fournir un journal de projet qui va démontrer le déroulement du développement de votre travail pratique. Voici que qu'il devrait comprendre.

- Page titre et table des matières.
- Un diagramme UML de base, avec seulement les classes et les relations entre elles.
- Description des assets : quel asset représente quoi dans le projet.
- Un journal de user stories :
  Pour chaque user story des sprints 1 à 4, vous inscrivez
  - Réalisé, réalisée partiellement, non réalisée
  - Qui l'a réalisé
  - Temps nécessaire (environ)
  - Si vous l'avez réalisé sans problème ou si des difficultés se sont posés

  Faites de même pour les stories des sprints 5 à 7, mais seulement si vous avez travaillé dessus.
- Un log de tâche : à chaque fois qu'un des membres de l'équipe travail sur le projet durant une journée donné, il doit inscrire le temps de travail total et le travail réalisé. Oui, ça peut recouper un peu le journal de stories, mais ici on est orienté sur le temps alors que de l'autre côté, on est orienté sur les stories.
- Ce journal est absolument obligatoire. Ne pas le remettre entraine une pénalité.
- Faites-le à mesure que le projet avance.
- **Je me répète, faites-le à mesure.**
- **Juste pour être absolument certain : FAITES-LE À MESURE**

CONTEXTE DE RÉALISATION ET DÉMARCHE DE DÉVELOPPEMENT

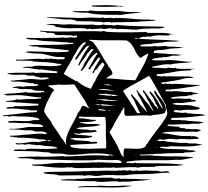
Ce travail pratique doit être réalisé **en équipe de 2.**

Biens livrables :

- Projet avec code source de l'application et journal de projet.
- Application fonctionnelle.
- Sur LEA pour **le dimanche 15 novembre à minuit.**

MODALITÉS D'ÉVALUATION

- Tous les **biens livrables** devront être **remis à temps** et selon les modalités spécifiées.
- Dans l'éventualité où vous récupériez du code existant ailleurs (internet, MSDN, etc), vous devez clairement indiquer la source ainsi que la section de code en question. Tout travail plagié ou tout code récupéré d'une source externe et non mentionnée peut entraîner la note zéro (0) pour l'ensemble du travail.



Grille d'évaluation du travail pratique #2

Critères d'évaluation	
Respect des contraintes générales de programmation	**15%**
Sprint 1 : Réalisation et respect des contraintes de programmation	**15%**
Sprint 2 : Réalisation et respect des contraintes de programmation	**15%**
Sprint 3 : Réalisation et respect des contraintes de programmation	**20%**
Sprint 4 : Réalisation et respect des contraintes de programmation	**10%**
Utilisation adéquate des concepts d'héritage et de polymorphisme (ex : mouvement partagé par tous les robots, tirs par tous les shooters, subir l'explosion par tous les personnages, etc.). Utilisation adéquate du Singleton et de la fabrique	**15%**
Journal de projet (zéro automatique si considéré comme "botché", -10% si non-remis). **Faites-le à mesure que le projet avance.**	**10%**
Éléments du Sprint 5 à 8 : Bonus discrétionnaires (jusqu'à 5% par story, selon la complexité) Minimum de 80% sur la base et remise du journal pour être considéré.	**Jusqu'à 5% par story**
TOTAL	**100%**