LEE Nathan
BAUDOT Victor
ABOUZAID Mehdi


# Projet Informatique

*Création d'un jeu – BLOC*

Mme.CABOT

Groupe STPI-2-K : 2016/2017

## Table des matières

# I. Description du projet

## 1. Cahier des charges

Titre du projet : BLOC

Descriptif détaillé :

BLOC est un jeu de plateforme dont l'objectif est de survivre le plus longtemps possible afin de gagner un maximum de points.
En démarrant, le jeu, le joueur doit donner son pseudo afin de charger ses données (meilleur score pour chaque personnage).
Le menu s'affiche alors avec les 4 différents personnages et le meilleur score réalisé pour chacun d'eux. Après avoir cliqué ou sélectionné le personnage avec les touches directionnelles du clavier et en ayant validé avec la touche BACKSPACE ou avec la souris, le jeu se lance.
Pour survivre, il faut déplacer son personnage avec les flèches gauche et droite du clavier afin d'éviter les blocs qui tombent et qui deviennent ensuite de nouvelles plateformes. Si on se trouve sous un bloc, on est aplati et l'écran « GAME OVER » apparaît. Pour escalader ou descendre les plateformes créées par les blocs, il suffit de maintenir la touche gauche ou droite du clavier et le personnage grimpera ou descendra. Il grimpera cependant plus lentement les blocs qu'il n'en tombera. La vitesse de descente des blocs augmentant au fur et à mesure de la partie, le niveau de difficulté s'accentuera. Des points d'exclamation apparaîtront en haut de l'écran à l'endroit où apparaîtront peu après les nouveaux blocs.

Kirby : Vitesse de déplacement normale. Pas de capacité spéciale.
Batman : Vitesse de déplacement normale. Il plane en descendant les blocs.
Hulk : Vitesse de déplacement réduite. Il possède deux vies.
Flash : Vitesse de déplacement augmentée. Pas de capacité spéciale.

Le score défile au cours du temps et est ensuite sauvegardée en fin de partie.

## 2. Liste des fonctionnalités attendues

- Initialisation du tableau du jeu (interface graphique)
- Création d’un personnage
- Déplacement du personnage au sein du jeu
- Générer des blocs
- Interaction personnage-blocs
- Déplacement des blocs
- Organisation des blocs (superposition et transformation en plateformes solides)
- Arrivée des blocs (points d'exclamations)
- Création de 3 personnages supplémentaires avec différentes caractéristiques
- Fin de partie (Écran Game Over)
- Connexion à son compte et chargement des scores
- Ajout des musiques de jeu et de fin de partie

## 3. Versions prévues (indiquant le contenu du programme dans cette version et sa date de livraison) :

- Version n°1, date prévue : 27/09
  fonctionnalités implémentées :
  - Initialisation du tableau du jeu (interface graphique)
  - Création d’un personnage
  - Déplacement du personnage au sein du jeu

- Version n°2, date prévue : 4/10
  fonctionnalités implémentées :
  - Générer des blocs
  - Déplacement des blocs

- Version n°3, date prévue : 18/10
  fonctionnalités implémentées :
  - Arrivée des blocs
  - Organisation des blocs
  - Interaction personnage-blocs

- Version n°4, date prévue : 15/11
  fonctionnalités implémentées :
  - Création de 3 autres personnages

- Fin de partie (Écran Game Over)
- Connexion à son compte et chargement des scores
- Ajout des musiques de jeu et de fin de partie

## 4. Signature et rôles des sous-fonctions

procedure **alerterNouveauBloc**(var Tableaujeu : Scenejeu);
→ Affiche une image d'alerte à l'emplacement où chaque nouveau bloc va apparaître

procedure **remplacerAlertesParBlocs**(var Tableaujeu : Scenejeu);
→ Remplace toutes les images d'alerte par un bloc

procedure **deplacerBlocs**(var Tableaujeu : Scenejeu; perso : String; var vie : Integer; var mort : boolean);
→ Si il y a une case vide sous un bloc, celui-ci descend à la case du dessous et ainsi de suite

procedure **verifierVideSousPerso**(p_scene : PScene; var tableau_jeu : Scenejeu);
→ Le personnage descend d'une case tant qu'il n'y a pas de bloc sur la case dessous lui

procedure **effacerDerniereLigne**(var Tableaujeu : Scenejeu; p_scene : PScene);
→ Efface la dernière ligne du tableau lorsqu'elle est pleine de blocs

procedure **lireEvenementClavier(**key : TSDL_KeyboardEvent; p_scene : PScene; perso : string; var tableau_jeu : Scenejeu);
→ Récupère la touche du clavier utilisé et déplace le personnage en conséquence (gauche, droite)

procedure **choisirPersonnage**(var p_screen : PSDL_SURFACE; p_sprite_sheet : PSpriteSheet; key : TSDL_KeyboardEvent; var choisi_personnage : Boolean; var perso : string; var i : Integer);
→ Choix du personnage avec les touches du clavier

procedure **lireEvenementSouris**(mouseEvent : TSDL_MouseButtonEvent; var choisi_personnage : Boolean; var perso : string);
→ Choix du personnage avec la souris

procedure **initialiser**(var Tableaujeu : Scenejeu);
→ Crée une ligne de bloc non visible à l'écran dans la ligne du tableau sous le personnage et initialise le tableau en le remplissant de 0.

procedure **lancerMusique**(musique : string);
→ Procédure de mise en musique

function **newSpriteSheet**() : PSpriteSheet;
→ Chargement des textures en mémoire

function **newScene**() : PScene;
→ Initialisation d'une scène

procedure **libererMemoire**(p_scene : PScene; p_sprite_sheet : PSpriteSheet);
→ Déchargement de la mémoire utilisée pour la scène et les textures

procedure **ecrire**(p_screen : PSDL_Surface; txt : String; x, y, taille : Integer);
→ Permet d'ajouter du texte à l'écran (scores, pseudonymes)

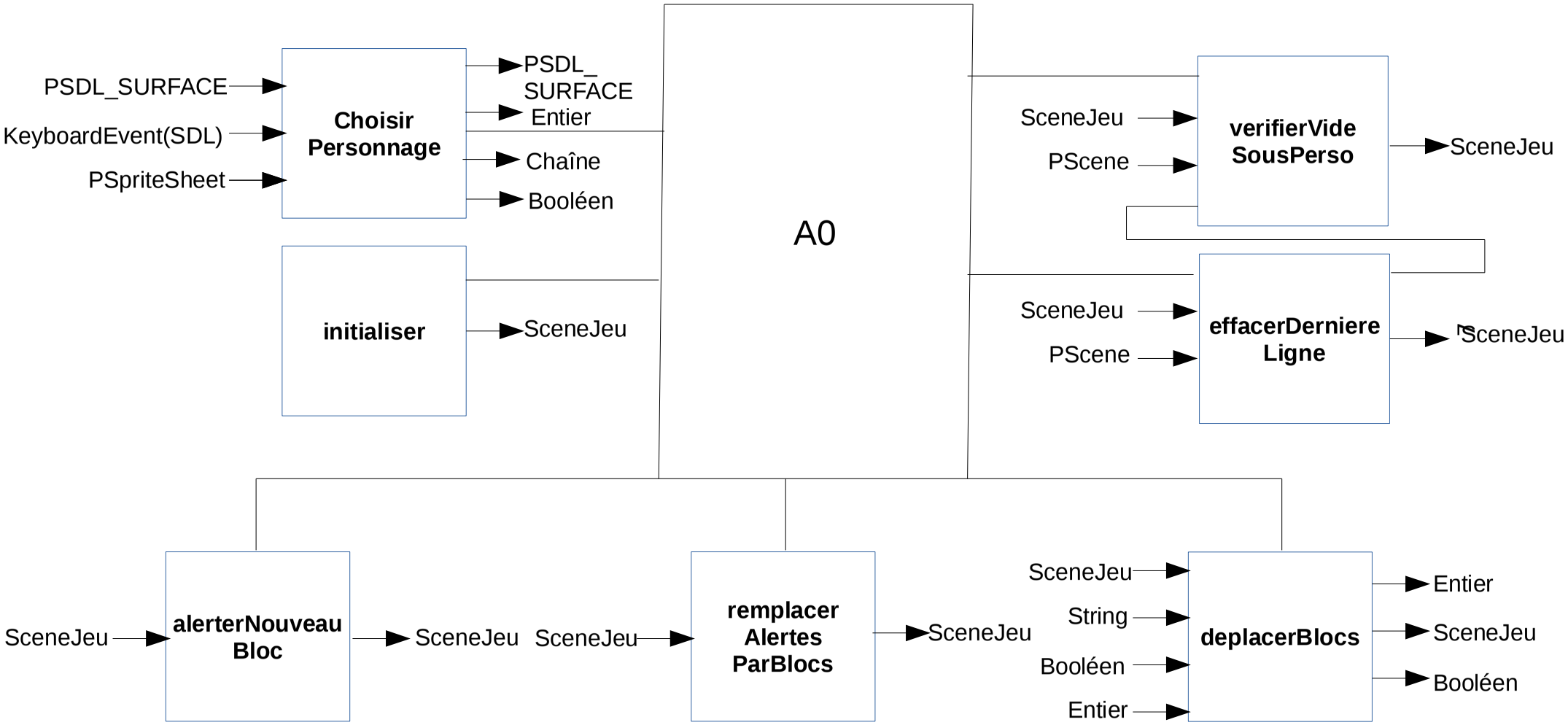
procedure **afficherEcranGameOver**(var p_screen : PSDL_SURFACE;
p_sprite_sheet : PSpriteSheet; score : longint);
→ Affiche l'écran (image) Game Over

procedure **afficherEcranMenu**(var p_screen : PSDL_SURFACE; p_sprite_sheet :
PSpriteSheet; mort : boolean; score : longint);
→ Affiche le fond et les différents personnages à l'écran

procedure **afficherAlertesBlocsPerso**(p_screen : PSDL_Surface;
p_sprite_sheet : PSpriteSheet; p_scene : PScene; perso : string; viesDeHulk :
Integer; var tableau_jeu : Scenejeu);
→Affiche les blocs et les points d'exclamation selon les valeurs de chaque case du tableau ainsi que le personnage

## 5. Analyse descendante

L'analyse descendante ci-dessous est différente de celle donnée au début du projet. En effet, au cours du projet nous avons augmenté le nombre de fonctionnalités (enregistrement des scores des joueurs, une page gameover, différents personnages etc...). Le passage sous SDL nous a obligé à ajouter un certains nombres de fonctions et procédures et nous a fait rencontré un certain nombre de problèmes que nous avons pu résoudre en modifiant notre programme mais cela nous a écarté fortement de l'analyse descendante initiale.
Toutes les procédures et fonctions sont maintenant appelés par A0 et il n'y a pas d'imbriquement de fonctions et procédures dans le programme si on écarte les fonctions et procédure de la SDL.


PSDL_SURFACE
KeyboardEvent(SDL)
PSpriteSheet
Choisir Personnage
PSDL_ SURFACE
Entier
Chaîne
Booléen
initialiser
SceneJeu
A0
SceneJeu
PScene
verifierVide SousPerso
SceneJeu
SceneJeu
PScene
effacerDerniere Ligne
SceneJeu
SceneJeu
alerterNouveau Bloc
SceneJeu
SceneJeu
SceneJeu
remplacer Alertes ParBlocs
SceneJeu
SceneJeu
String
Booléen
Entier
deplacerBlocs
Entier
SceneJeu
Booléen

## II. Guide d’utilisation

Après avoir entré son pseudo, un menu se présente avec 4 personnages au choix : Kirby (rose), Flash (jaune), Hulk (vert) et Batman (rouge), avec des fonctionnalités différentes. Kirby se déplace normalement; Hulk se déplace lentement et a une vie supplémentaire; Batman descend en “volant” et Flash se déplace rapidement. Les records obtenus avec chacun de ces personnages sont également affichés.
Après avoir sélectionné un des personnages, soit avec la souris, soit avec les flèches du clavier (et backspace pour valider), le jeu se lance. Il faut déplacer le personnage, ce qui est possible uniquement avec les flèches Gauche et Droite du clavier, tout en évitant les blocs qui descendent. Lorsque les blocs atteignent le sol, le personnage peut monter dessus pour continuer à se déplacer de gauche à droite. Le personnage meurt dès qu’un bloc l’a heurté. L'écran Game Over s'affiche alors avec le score du joueur. Le joueur peut cliquer avec la souris ou appuyer sur la touche espace pour retourner au menu principal. Le joueur peut alors rejouer avec l'un des différents personnages.

## III. Perspectives d’évolution

Pour notre jeu BLOC, nous avons pensé à différentes perspectives d’évolution :

- Rajouter des bonbons qui tombent aléatoirement selon le temps. Ils peuvent soit ajouter une vie, soit en enlever une, appliquer un malus ou un bonus sur le score et ralentir ou accélérer la descente des blocs.
- L’ajout d’un bouton pause ou la possibilité d’appuyer sur une touche pour mettre le jeu en pause. Bien sûr l’écran se voilerait pour empêcher l’utilisateur de « tricher ».
- Mettre de nouvelles structures de jeu en place : changer les règles pour contraindre le joueur à s’adapter à de nouvelles situations (ex : mettre une hauteur maximale pour monter, ou une hauteur à atteindre, avec de la lave qui monte).
- Choix de la texture des blocs et du fond d’écran du jeu
- Pouvoir jouer à plusieurs sur le même ordinateur en donnant des touches différentes pour chaque joueur.

# IV. Organisation

## 1. Planning

Le planning que nous avions établi dans le cahier des charges a bien été respecté et nous avons même pu ajouter des fonctionnalités non prévues au départ. A chaque fois que nous avions une nouvelle idée, nous la rajoutions dans le cahier des charges sans sur-estimer nos capacités à les créer à temps.

## 2. Répartition des tâches

**Mehdi** : score, fichier score, alerter l'arrivée des blocs et les générer, musiques sous sdl

**Victor**: affichage du jeu sous sdl, menu principal, lecture des événements au clavier et à la souris pour le choix du personnage, initialiser le tableau, procédure VerifierVideSousPerso

**Nathan** : écriture du programme sous sdl, procédure effacerDerniereLigne, déplacement des blocs

Mehdi s’est occupé à mettre en place le score, et a fait en sorte que celui-ci augmente au fil du temps, en le plaçant dans la boucle principale. Pour l’afficher, Nathan a trouvé une fonction SDL permettant d ‘écrire dans la scène de jeu : nous nous en sommes servi pour afficher le score ainsi que les records établis par des joueurs précédents, avec leur pseudonyme à côté. En effet Mehdi a créé un fichier pouvant contenir les scores, et les joueurs attachés. Il a fait en sorte que le fichier contienne un tableau des joueurs, avec tout ce qu’il fallait à l’intérieur. Il s’est aussi occupé de la musique sous SDL, et a ainsi ajouté deux musiques, une pour le jeu, et une pour l’écran gameover. Il a aussi élaboré la fonction remplacerAlertesParBlocs, et l’a placé ainsi que la fonction alerterNouveauBloc, qu’il a lui aussi conçu, dans le programme principal. Il a réussi à limiter les actions de ces deux fonctions dans la boucle principale, lors de la structuration du jeu.

Nathan quant à lui, a élaboré la fonction effacerDerniereLigne, celle-ci servant à effacer la dernière ligne du tableau pour éviter un entassement de blocs trop important. Quand il l'a créé, il n'avait pas encore inclus la fonction verifierVideSousPerso : ce n'est que lorsque nous avons mis tout ensemble que

nous avons réalisé sa nécessité (expliqué dans difficultés). Il a aussi mis en place la fonction deplacerBloc, et l'a installé dans la boucle principale et Victor s'est occupé de la vitesse d'apparition des alertes et des blocs. Nathan s'est finalement occupé de la fonction lireEvenementClavier, et a réussi à gérer les différents personnages et leurs différents déplacements.

Victor, lui, a créer tout l'affichage SDL, Il s'est donc occupé de la plus grande partie de l'intégration à la SDL. Il a constitué la fonction ChoisirPersonnage, pour permettre à l'utilisateur de choisir un personnage de manière sympathique, ainsi que la fonction lireEvenementSouris, celle-ci allant de pair avec l'autre. Il a aussi initialisé le tableau avec la fonction initialiser. Et il a composé la fonction verifierVideSousPerso, vitale pour empêcher le personnage de rester en l'air. Il s'est également chargé de faire en sorte que la vitesse de descente des blocs augmente progressivement et que la vitesse de déplacement des personnages puisse être ajuster.

## 3. Difficultés

### *Adaptation des coordonnées du tableau*

Nous avons eu des problèmes au niveau de l’adaptation sous SDL : l'un des problèmes majeurs a été la retranscription de la position réelle du personnage dans le tableau pour qu’il puisse bouger correctement et interagir avec les blocs. En effet, nous avons utilisé Pscene pour le support du personnage sous la SDL, et grâce à p_Scene.i et p_Scene.j, la position du personnage dans la scène de jeu nous était transmise. Cependant celle-ci n’était pas correcte par rapport à notre tableau de jeu : on pouvait retrouver « 3 » en coordonnée x transmise alors que le personne était visiblement en coordonnée « 2 » dans la scène du jeu. Après plusieurs tests, nous nous sommes rendus compte que p_Scene.i et p_Scene.j, qui sont les positions de notre personnage, n'affichaient pas les bonnes coordonnées. En effet, la position renvoyée par ces variables correspond à l'angle haut-gauche du perso. Donc en modifiant par i+1 et j+1, l'image affiché correspond à sa position dans le tableau et faire en sorte qu’il interagisse correctement avec les blocs.

### *Descente des blocs*

Pour la descente des blocs, les problèmes ont d’abord été au niveau de la boucle les permettant de descendre : en effet nous parcourions le tableau de gauche à droite, et de haut en bas. De ce fait quand nous déplacions un bloc,

nous le re-déplacions juste après, et donc le bloc se retrouvait tout en bas en un seul appel de la fonction deplacerBlocs, ce qui n'était pas ce que l'on voulait. Nous avons donc fait en sorte de parcourir la boucle en sens inverse pour empêcher de re-déplacer des blocs déjà déplacés. Ainsi, deplacerBlocs ne déplace que les blocs d'une case en coordonnée y, c'est ce que l'on voulait.

### *Interaction personnage-blocs*

Le personnage ne doit pas rester en l'air quand il se déplace ou quand une ligne remplie s'efface. Nous avons donc pensé à créer une fonction verifierVideSousPerso qui serait appelée à chaque tour de jeu et permettrait de faire tomber le personnage si rien ne se trouve en dessous de lui. Mais un gros problème est apparu : lorsque le personnage monte, du vide se trouve sous lui, donc cette fonction l'empêche de monter quoi qu'il fasse. Ainsi, il a fallu utiliser SDL_KEYUP pour savoir si le joueur n'est pas en train de monter, et si c'est le cas, le faire descendre. Malheureusement, cette implémentation a engendrée une nouvelle erreur : si le personnage garde une touche enfoncée, alors lorsque l'on effaçait une ligne, il pouvait rester en l'air. Nous avons du faire en sorte d'appeler la fonction verifierVideSousPerso dans cette procédure également pour que le personnage descende correctement quand il le faut.

### *Vitesse et déplacement du personnage*

Au niveau de la gestion des touches appuyées par le joueur et le déplacement du personnage en réponse, nous avons rencontré un autre problème : il y avait un léger lag au niveau de la réponse aux touches appuyées par le joueur, et il arrivait que celui-ci n'arrivait pas à bouger même si une flèche du clavier a été appuyée. Après expérimentation, nous avons réalisé que le SDL_Delay était responsable du problème. Nous avions mis celui-ci dans notre boucle pour s'assurer que le personnage ne puisse pas bouger trop vite. Or si le joueur appuyait sur une touche alors que le SDL_Delay était actif, le personnage ne pouvait pas bouger. Nous l'avons donc supprimé mais avons vite compris que le personnage pouvait maintenant se déplacer aussi vite qu'une nouvelle boucle de jeu commençait : c'est à dire 1 déplacement par milliseconde (environ). Pour régler ce problème, nous avons eu l'idée de créer une variable, qui se réinitialisera à chaque fois que le joueur tente de bouger, que ce soit correctement ou incorrectement (touches aléatoires se trouvant sur le clavier). Cette variable doit être ensuite augmenter en valeur suffisante pour qu'on accepte une nouvelle action de la part du joueur, s'assurant ainsi que le personnage ne puisse pas se déplacer trop rapidement.

### *Initialisation tableau et position du personnage*

Au niveau de l’initialisation, nous avons pensé que la position du personnage pouvait être initialisée avec le tableau de jeu au même moment mais cela a conduit à un effacement du personnage avant même que le joueur puisse le bouger. Il apparaissait à nouveau lorsque le joueur appuyait sur une touche. Nous avons donc initialisé la position du personnage en dehors de cette procédure, et de ce fait, n’avons plus eu de problèmes.

## Conclusion

L'année dernière, nous avons découvert la programmation que ce soit en pseudo-code ou en Pascal mais ce n'est qu'au cours de ce projet que nous avons réellement pu appréhender les possibilités qu'offrait la programmation. Nous avons choisi de créer un jeu pour rendre le projet fun et essayer les interfaces graphiques, ici la SDL. De nombreux problèmes ont été rencontrés au cours de la programmation mais nous avons toujours trouvé une solution et c'est ce qui nous a motivé à ajouter toujours plus de fonctionnalités. Réussir à travailler en groupe, à se répartir les tâches n'est pas aisé mais c'est grâce à ce genre de projet et aux problèmes rencontrés que nous nous améliorons afin d'être plus efficaces et autonomes. Etant tous déterminés à rentrer au département ASI, nous avons hâte de nous lancer dans un nouveau projet encore plus ambitieux.