ÉCOLE NATIONALE D'INGÉNIEURS DE BREST

RÉVÉLATEUR D'INGÉNIEURS
DEPUIS 1961

Document de conception

BlindQuest-S2

Nom du projet	**BlindQuest-S2**
Version	1.0
Date	09/04/2015
Auteurs	CARON Clémence – c4caron@enib.fr GONI Guillaume – g4goni@enib.fr

1 Rappel du cahier des charges

1.1 Le jeu

BlindQuest est un jeu de rôle pour un joueur. Il faut se déplacer à travers un monde, franchir différents environnements et vaincre le « boss » final, en se laissant guider par la musique ambiante.

1.2 Contraintes techniques

- BlindQuest doit fonctionner sur les machines TP de l'ENIB, et prendre en charge les fonctionnalités audio.
- Le développement devra se faire en python.
- Nous utiliserons de la programmation objet, avec un raisonnement objet, mais un déroulement procédural pour rester en adéquation avec le cours de MDD.
- La barrière d'abstraction sera grandement employée.

1.3 Fonctionnalités

- F1 : Charger la carte
- F2 : Restituer l'environnement sonore
- F3 : Gestion des déplacements avec les touches [↑] [↓] [→] [←]
- F4 : Gestion des combats
  - F4.1 : Attaque du joueur (succès (60%) ou échec)
  - F4.2 : Gestion de la vie
- F5 : Gestion de la mort
  - F5.1 : Chute dans l'eau
  - F5.2 : Décès lors d'un combat
- F6 : Quitter le jeu
  - F6.1 : Victoire contre le boss final
  - F6.2 : Mort
  - F6.3 : Pression de la touche [Q]
- F7 : Gestion de la sauvegarde
  - F7.1 : Sauvegarde avec la touche [S]
  - F7.2 : Charger une sauvegarde avec [C]
- F8 : Mettre en pause en appuyant sur [SPACE]
- F9 : Afficher l'aide – les commandes – avec la touche [H]

1.4 P1 : Prototype P1

Ce prototype mettra en place le chargement de la carte F1, la restitution de différents environnements F2, le déplacement F3 et la possibilité de quitter le jeu F6.3.

Toutes les versions du jeu sont livrées dans une archive au format .zip ou .tgz et contiennent le manuel d'utilisation dans le fichier README.md

1.5 P2 : Prototype P2

Cette version sera complétée par des améliorations de l'environnement F5.1, notamment la création de milieux dangereux pour le joueur, les monstres et la fonction de combat F4, F6.1, F6.2, le décès du personnage F5, le mode pause F8, l'aide F9, et éventuellement la sauvegarde F7.

1.5 Version finale

Contient toutes les versions des documents : cahier des charges et conception, prototypes, manuel d'utilisation.

2 Principes des solutions techniques

2.1 Langage

Conformément aux contraintes énoncées dans le cahier des charges, le codage est réalisé en Python. Nous proposons une version 2.7 et une version 3.4.

2.2 Architecture du logiciel

Nous mettons en œuvre le principe de la barrière d'abstraction. Chaque module correspond à une classe et contient tous ses attributs et méthodes permettant de le manipuler de manière abstraite.

m_jeu : Contient la classe Jeu qui gère les interactions utilisateur – affichage et son.
m_carte : Contient la classe Carte qui gère la carte et les déplacements du joueur.
constantes : Fichier de configuration : informations du joueur, carte utilisée, etc.

Le jeu est disponible pour Mac, Linux (toutes distributions) et Windows.

2.3 Outils utilisés

On utilise les modules Pyglet et Time, ainsi que le module OS pour se déplacer dans les fichiers. Pour le son, la bilbiothèque Openal est nécessaire, nous avons également utilisé le gestionnaire de versions Github : github.com/guydunigo/Jeu-Python-S2.git

Un script est disponible pour faciliter le lancement de BlindQuest, toutes les options sont détaillées dans le fichier README ; une version de Pyglet est proposée au chargement dans le dossier du jeu.

2.4 Interface utilisateur

L'interface utilisateur se fait via une fenêtre en plein écran par défaut, gérée par Pyglet, et par l'intermédiaire de la sortie audio du système.

2.4.1 Boucle de simulation

Le programme mettra en œuvre la boucle de simulation Pyglet qui gérera le son, les événements clavier et l'affichage.

2.4.2 Son

Le son est transmis via la sortie audio de la machine, grâce au module Pyglet. Les fichiers musicaux seront stockés indépendamment.

2.4.3 Affichage

L'affichage se fait sur une fenêtre Pyglet en plein écran par défaut.

2.4.4 Gestion du clavier

Le module Pyglet est utilisé pour détecter les actions de l'utilisateur, il permet de rediriger les événements clavier.

2.5 Carte

Pour modéliser la carte, une liste de listes permet de stocker les valeurs correspondant aux différents environnements.

0 : Plaine
1 : Forêt
2 : Caverne
3 : Eau, rivière ☠
4 : Château
5 : Sentier de la forêt
6 : Pont
7 : Sable
8 : Montagne
9 : Forêt maudite
14 : Mer ☠
15 : Entrée château

10 : Monstre
11 : Boss
12 : Boss final
13 : Bonus

97 : Bordure de terrain
98 : Début de carte
99 : Fin de carte

3 Analyse

3.1 Analyse noms/verbes :

- Verbes :

Charger, restituer, déplacer, combattre, attaquer, gérer, mourir, quitter, vaincre, sauvegarder, afficher.

- Noms :

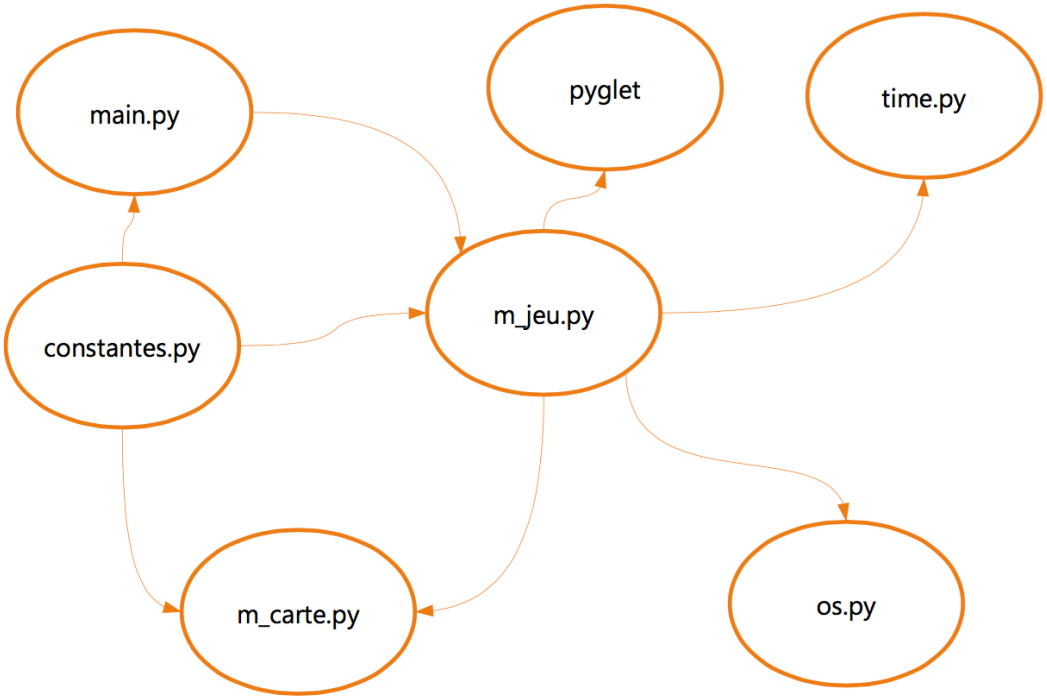
Carte, environnement sonore, touches, joueur, vie, eau, jeu, pause, aide.

3.2 Types de données

Type : jeu = struct
    carte : Carte
    vie : entier
    sons : dico(chaîne, pyglet.resource.media)
    lecteurs : dico(chaîne, pyglet.media.player)
    window : pyglet.window.Window
    state : chaîne de caractères
    pause : liste de chaînes de caractères
    text : liste [pyglet.text.Label]

Type : Carte = struct
    carte : liste de liste d'entiers
    position_joueur : liste [x,y]
    nb_lignes : entier
    nb_colonnes : entier
    player_info : liste

3.3 Dépendance entre modules



3.4 Analyse descendante

3.4.1 Arbre principal

```
main.py
+-- m_jeu.Jeu() #Crée l'objet jeu
|       +-- m_carte.Carte() #Crée l'objet carte
|       |       +-- ouvrir_fichier_carte()
|       |       |       +-- charger_carte()
|       |       +-- trouver_depart()
|       |               +-- carte._get_posx()
|       |               +-- carte._get_posy()
|       |               +-- carte._set_posx()
|       |               +-- carte._set_posy()
|       +-- charger_son()
|       +-- creer_lecteurs()
|       +-- creer_fenetre()
|       +-- init_events()
+-- jeu.run()
        +-- on_key_press()
```

3.4.2 Arbre interaction

```
on_key_press()
+-- jeu.afficher_aide()
```


c4caron@enib.fr - g4goni@enib.fr

```
+-- jeu.move()
|       +-- carte.empty()
|       +-- carte.move()
|       |       +-- carte._get_posx()
|       |       +-- carte._get_posy()
|       |       +-- carte._set_posx()
|       |       +-- carte._set_posy()
|       |       |
|       |       +-- carte.detect_prox()
|       |               +-- carte._get_posx()
|       |               +-- carte._get_posy()
|       |               +-- carte._set_posx()
|       |               +-- carte._set_posy()
|       +-- jeu.fin()
|       +-- jeu.combat_init()
+-- jeu.save()
|       +-- carte.save()
+-- jeu.load()
|       +-- m_carte.Carte()
|       |       +-- ouvrir_fichier_carte()
|       |               +-- charger_carte()
|       +-- get_player_info()
+-- jeu.pause()
+-- tour_combat()
        +-- jeu.fin()
        +-- jeu.attaque()
        +-- jeu.empty()
```

4 Description des fonctions

4.1 Classe Jeu : m_jeu.py

[En vert, les fonctions pour le second prototype]

- Jeu.**__init__(self, type_carte = « defaut »)**
- Jeu.**_get_vie()**
- Jeu.**_set_vie()**
- Jeu.**charger_sons(self)**
- Jeu.**creer_lecteurs(self)**
- Jeu.**creer_fenetre(self)**
- Jeu.**init_events(self)**
- Jeu.**move(self, direction = None)**
- Jeu.**afficher_aide(self)**
- Jeu.**fin(self, type_fin)**
- Jeu.**combat_init(self)**
- Jeu.**attaque(self, proba)**
- Jeu.**tour_combat(self)**
- Jeu.**run(self)**

- Jeu.**load(self)**
- Jeu.**save(self, player_infos)**
- Jeu.**afficher_load()**
- **on_key_press(symbol, modifiers)**
- **on_key_release(symbol, modifiers)**
- **on_draw()**
- Jeu.**pause()**

Jeu.**__init__(self, type_carte = « defaut »)** -> instance de la classe Jeu

Description : Constructeur de la classe Jeu
Paramètres : type_carte : chaîne de caractères non vide
Valeur de retour : Jeu

Jeu.**_get_vie(self)** -> entier

Description : Renvoie la vie du joueur
Paramètres : aucun
Valeur de retour : entier

Jeu.**_set_vie(self, new_vie)** -> rien

Description : Modifie la vie du joueur, lance le son « battements de cœur » pour une vie inférieure à 3.
Paramètres : new_vie : entier (nouvelle valeur correspondant à la vie du joueur)
Valeur de retour : rien

Jeu.**charger_sons(self)** -> rien

Description : Charge les sons à partir du dossier sons
Paramètres : aucun
Valeur de retour : rien

Jeu.**creer_lecteurs(self)** -> rien

Description : Crée les lecteurs pyglet ; les lecteurs env, eau, et heartbeat sont réglés pour tourner en boucle sur la musique en cours.
Paramètres : aucun
Valeur de retour : rien

Jeu.**creer_fenetre(self)** -> rien

Description : Crée la fenêtre Pyglet en plein écran.
Paramètres : aucun
Valeur de retour : rien

Jeu.**init_events(self)** -> rien

Description : Crée les événements claviers
Paramètres : aucun
Valeur de retour : rien

Jeu.**move(self, direction = None)** -> rien

Description : Fonction qui déplace le joueur et gère ce qui peut arriver, lance les sons d'environnement et de proximité.

Paramètres : directions : chaîne de caractères non vide

Valeur de retour : rien

Jeu.**afficher_aide(self)** -> rien

Description : fonction qui affiche l'aide

Paramètres : aucun

Valeur de retour : rien

Jeu.**fin(self, type_fin)** -> rien

Description : Fonction qui gère la fin du jeu (victoire/mort), quitte le programme

Paramètres : type_fin : chaîne de caractères

Valeur de retour : rien

Jeu.**combat_init(self)** -> rien

Description : Fonction qui permet de lancer les combats

Paramètres : aucun

Valeur de retour : rien

Jeu.**attaque (self, proba)** -> dégâts

Description : Fonction qui gère l'attaque avec le pourcentage de réussite

Paramètres : proba : entier

Valeur de retour : dégâts : entier

Jeu.**tour_combat (self)** -> rien

Description : Gère le tour du combat : attaque du joueur, et du monstre

Paramètres : aucun

Valeur de retour : rien

Jeu.**run(self)** -> rien

Description : Lance la partie

Paramètres : aucun

Valeur de retour : rien

Jeu.**load(self)** -> rien

Description : Charge une partie sauvegardée

Paramètres : aucun

Valeur de retour : rien

Jeu.**save(self, player_infos)** -> rien

Description : Sauvegarde la partie dans un fichier.

Paramètres : player_infos : liste contenant les informations du joueur

Valeur de retour : rien

Jeu.**afficher_load(self)** -> rien

Description : Affiche les sauvegardes qui peuvent être chargées
Paramètres : aucun
Valeur de retour : rien

On_key_press (symbol, modifiers) -> rien
Description : Effectue une action lorsqu'une touche est appuyée
Paramètres : modifiers, symbol : touche du clavier pyglet
Valeur de retour : rien

On_key_release(symbol, modifiers) -> rien
Description : Effectue une action lorsqu'une touche est relâchée
Paramètres : modifiers, symbol : touche du clavier pyglet
Valeur de retour : rien

On_draw() -> rien
Description : Gère l'affichage
Paramètres : aucun
Valeur de retour : rien

Jeu.pause() -> rien
Description : Met le jeu en pause
Paramètres : aucun
Valeur de retour : rien

4.2 Classe Carte : m_carte.py

- Carte.**__init__()**
- Carte.**__repr__()**
- Carte.**_get_posx(self)**
- Carte.**_get_posy(self)**
- Carte.**_set_posx(self, new_posx)**
- Carte.**_set_posy(self, new_posy)**
- Carte.**_get_case()**
- Carte.**_set_case()**
- Carte.**_get_nb_colonnes()**
- Carte.**_set_nb_colonnes()**
- Carte.**_get_nb_lignes()**
- Carte.**_set_nb_lignes()**
- Carte.**get_case_type()**
- Carte.**set_case_type()**
- Carte.**ouvrir_fichier_carte(self, dossier, nom_fichier)**
- Carte.**charger_carte(self, fichier_carte)**
- Carte.**get_player_info(self)**
- Carte.**trouver_depart(self)**
- Carte.**move(self, direction = None)**
- Carte.**detect_prox(self)**
- Carte.**save(self, nom_fichier, player_info)**

Carte.**empty(self)**

Carte.**__init__(self, type_carte = « defaut », num_sauv = None**) -> objet de type carte

Description : Constructeur : Charge la carte dans une liste de liste (attribut carte) et défini la position par défaut du joueur (attributs posx et posy).

Paramètres : type_carte : chaîne de caractères non vide
Num_sauve : int ou chaîne de caractères

Valeur de retour : Carte

Carte.**__repr__(self)** -> chaîne de caractères

Description : Renvoie la carte sous forme de chaîne de caractères, et l'affiche dans le terminal

Paramètres : aucun

Valeur de retour : chaîne de caractères

Carte.**_get_posx(self)** -> entier

Description : Accesseur de l'attribut posx

Paramètres : aucun

Valeur de retour : entier

Carte.**_get_posy(self)** -> entier

Description : Accesseur de l'attribut posy

Paramètres : aucun

Valeur de retour : entier

Carte.**_set_posx(self, new_posx)** -> rien

Description : Mutateur de l'attribut posx, on ne peut affecter une abscisse invalide (ex : sortir de la carte, montagne…)

Paramètres : new_posx : entier

Valeur de retour : rien

Carte.**_set_posy(self, new_posy)** -> rien

Description : Mutateur de l'attribut posy

Paramètres : new_posy : entier

Valeur de retour : rien

Carte.**_get_case(self)** -> entier

Description : Renvoie le type de case sur laquelle le joueur se trouve

Paramètres : aucun

Valeur de retour : entier

Carte.**_set_case(self, new)** -> rien

Description : Modifie le type de case sur laquelle le joueur se trouve

Paramètres : new : nouvelle valeur affectée

Valeur de retour : rien

Carte.**_get_nb_colonnes(self)** -> entier

Description : Renvoie le nombre de colonnes de la carte
Paramètres : aucun
Valeur de retour : entier

Carte.**_set_nb_colonnes(self, new)** -> rien
Description : Bloque la modification du nombre de colonnes de la carte
Paramètres : new : nouvelle valeur affectée
Valeur de retour : rien

Carte.**_get_nb_lignes(self)** -> entier
Description : Renvoie le nombre de lignes de la carte
Paramètres : aucun
Valeur de retour : entier

Carte.**_set_nb_lignes(self, new)** -> rien
Description : Bloque la modification du nombre de lignes de la carte
Paramètres : new : nouvelle valeur affectée
Valeur de retour : rien

Carte.**get_case_type(self, x, y)** -> entier
Description : Renvoie le type de la case spécifiée en x et y
Paramètres : x et y entiers (coordonnées sur la carte)
Valeur de retour : entier

Carte.**set_case_type(self, x, y, new_type)** -> rien
Description : Affecte un nouveau type d'environnement à la case spécifiée
Paramètres : x, y, et new_type (nouvelle variable d'environnement) entiers
Valeur de retour : rien

Carte.**ouvrir_fichier_carte(self, dossier, nom_fichier)** -> rien
Description : Charge la carte du fichier nommé 'nom_fichier.txt', présent dans le dossier donné (classiquement saves ou cartes), sous la forme d'un liste de listes d'entiers.
Paramètres : dossier : chaîne de caractères ; nom_fichier : chaîne de caractères non vide correspondant à un dossier existant
Valeur de retour : rien

Carte.**charger_carte(self, fichier_carte)** -> rien
Description : Charge le contenu d'un fichier carte dans une liste de liste de int, si les lignes ne sont pas toutes de la même taille, on les complète par de l'eau.
Paramètres : fichier_carte : fichier
Valeur de retour : rien

Carte.**get_player_info(self)** -> rien
Description : Renvoie la liste d'informations du joueur lors du chargement d'une sauvegarde, sans les informations concernant la position.
Paramètres : aucun
Valeur de retour : rien

Carte.**trouver_depart(self)** -> rien

Description : Trouve les coordonnées du départ et les affecte à posx et posy.
Paramètres : aucun
Valeur de retour : rien

Carte.**move(self, direction = None)** -> entier ou None

Description : Déplace le joueur et renvoie le code de la case d'arrivée.
Paramètres : direction : None ou chaîne de caractères
Valeur de retour : entier ou None

Carte.**detect_prox(self)** -> entier

Description : Renvoie un entier composé au maximum de 4 puissances de 2 différentes additionnées pour l'utilisation de l'opérateur bit à bit.
Paramètres : aucun
Valeur de retour : entier

Carte.**save(self, nom_fichier, player_info)** -> rien

Description : Sauvegarde la partie dans un fichier.
Paramètres : nom_fichier : chaîne de caractères non vide
player_info : liste
Valeur de retour : rien

Carte.**empty (self)** -> entier

Description : affecte à la case actuelle la valeur de la case adjacente (si elle est assignable)
Paramètres : aucun
Valeur de retour : entier (valeur de la nouvelle case)

5 Calendrier et suivi de développement

5.1 Prototype 1

5.1.1 Fonctions à développer

Fonctions	Codées	Testées	Commentaires
Jeu.**__init__(self, type_carte = « defaut »)**			
Jeu.**_get_vie()**			
Jeu.**_set_vie()**			
Jeu.**charger_sons(self)**			
Jeu.**creer_lecteurs(self)**			
Jeu.**creer_fenetre(self)**			
Jeu.**init_events(self)**			
Jeu.**move(self, direction = None)**			
Jeu.**fin(self, type_fin)**			

Jeu.**run(self)**			
on_key_press(symbol, modifiers)			
Carte.**__init__()**			
Carte.**_get_posx(self)**			
Carte.**_get_posy(self)**			
Carte.**_set_posx(self, new_posx)**			
Carte.**_set_posy(self, new_posy)**			
Carte.**_get_case()**			
Carte.**_set_case()**			
Carte.**_get_nb_colonnes()**			
Carte.**_set_nb_colonnes()**			
Carte.**_get_nb_lignes()**			
Carte.**_set_nb_lignes()**			
Carte.**get_case_type()**			
Carte.**set_case_type()**			
Carte.**ouvrir_fichier_carte(self, dossier, nom_fichier)**			
Carte.**charger_carte(self, fichier_carte)**			
Carte.**trouver_depart(self)**			
Carte.**move(self, direction = None)**			
Carte.**detect_prox(self)**			
Carte.**save(self, nom_fichier, player_info)**			

5.1.2 Autres

Dossier sons, dossier cartes (avec carte_petite), dossier src (4 fichiers python), README.md, blindquest.sh

5.2 Prototype 2

5.2.1 Fonctions à développer

Fonctions	Codées	Testées	Commentaires
Jeu.**afficher_aide(self)**			
Jeu.**load(self)**			
Jeu.**save(self, player_infos)**			
Jeu.**afficher_load()**			
on_key_release(symbol, modifiers)			
on_draw()			
Carte.**get_player_info(self)**			
Jeu.**pause()**			
Jeu.**combat_init(self)**			
Jeu.**attaque(self, proba)**			
Jeu.**tour_combat(self)**			
Carte.**empty(self)**			

5.2.2 Autres

Carte complète, tous les sons.