

Développement Logiciel

L2-S4

Présentation Projet
2014

anastasia.bezerianos@lri.fr

Objectif du Projet (en binome)

- Le projet s'étend sur les **6 dernières séances de TP**
- Le projet se découpe en **2 sous-parties (version initiale et avancée)**
- Evaluation (*attention: il faut être prêt pendant les TPs*).
 - à la 3ème séance, une **soutenance intermédiaire** sur machine
 - vous devez rendre aussi:
 - un mini-rapport
 - » décrivant les fonctionnalités de votre code et vos responsabilités
 - » montrant une copie d'écran pour chaque sous-partie
 - votre code
 - » les sources
 - » un fichier aide.txt pour compiler, lancer
 - à la dernière séance, une **soutenance finale** sur machine
 - (idem)
 - vos observations sur les paramètres de la simulation

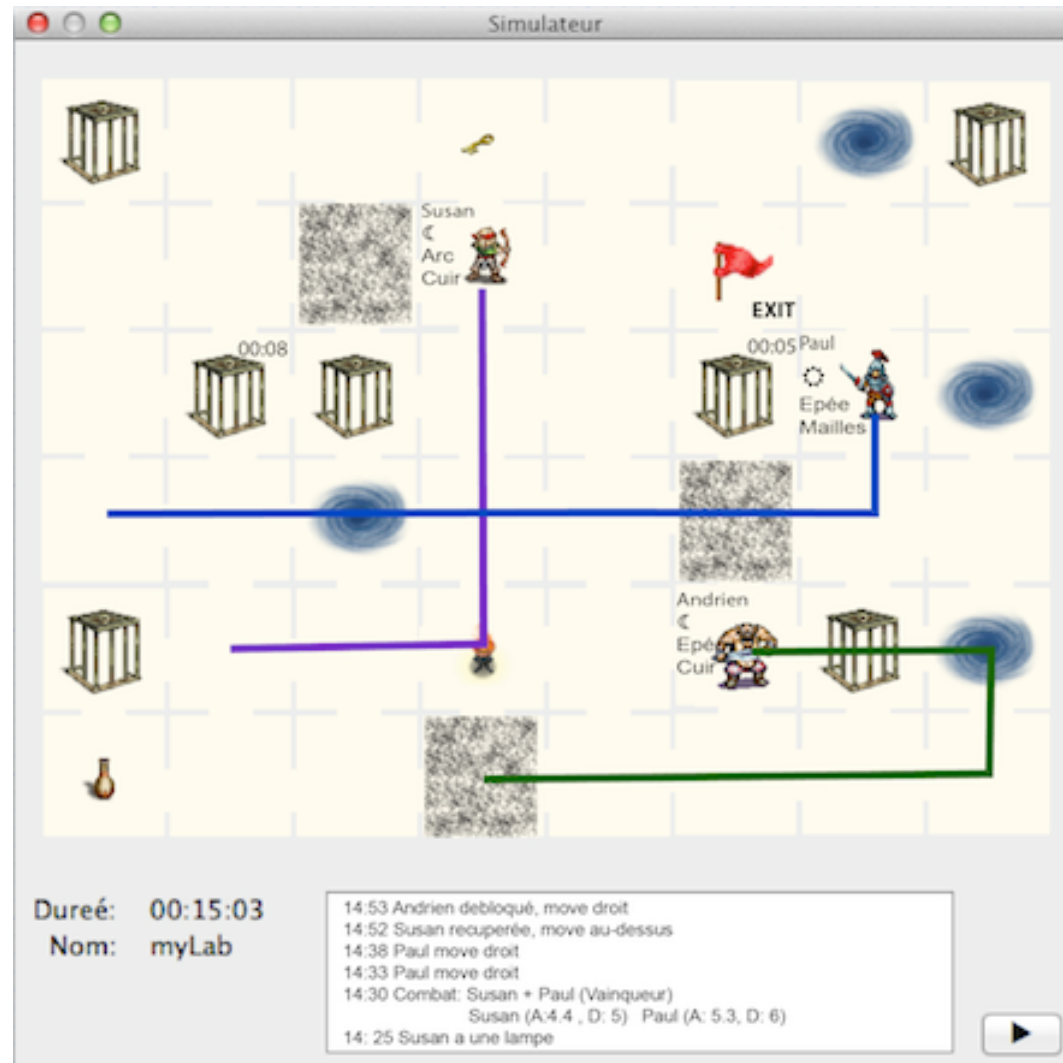
Sujet du projet : un labyrinthe

programmer un jeu de labyrinthe, où l'objectif des personnages est de sortir du labyrinthe.

Trois parties:

1. **L'atelier** : permet la conception des types de personnages pour notre jeu
2. **L'éditeur de labyrinthe** : permet la conception d'un labyrinthe (grille à deux dimensions de salles)
3. **Le simulateur de jeu** : simule l'avancement des personnages dans le labyrinthe jusqu'à leur sortie

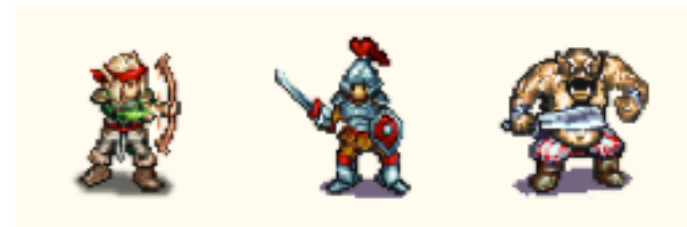
Sujet du projet : un labyrinthe



Sujet du projet : un labyrinthe

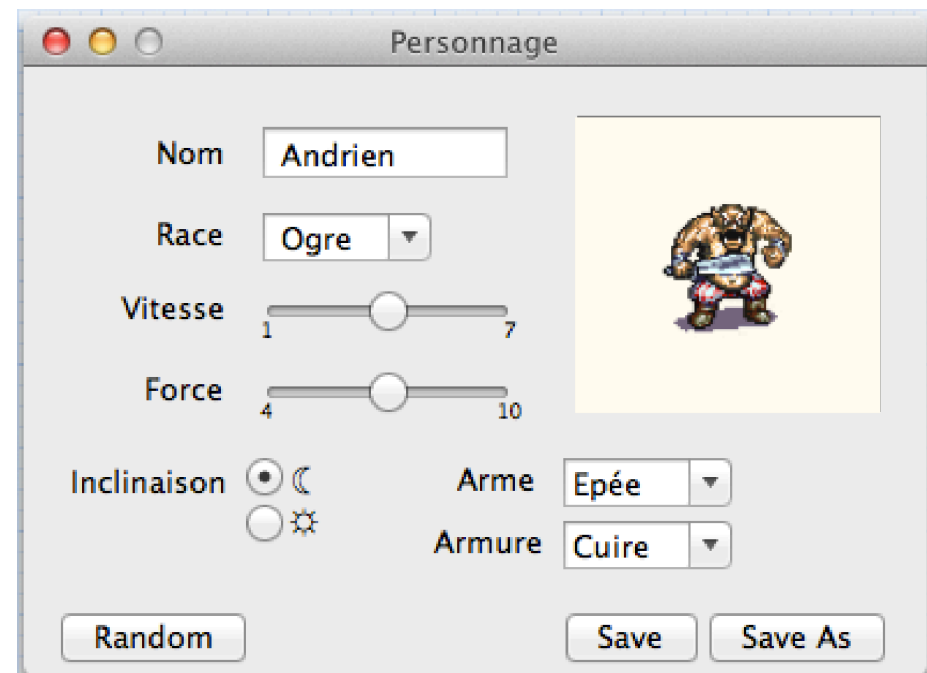
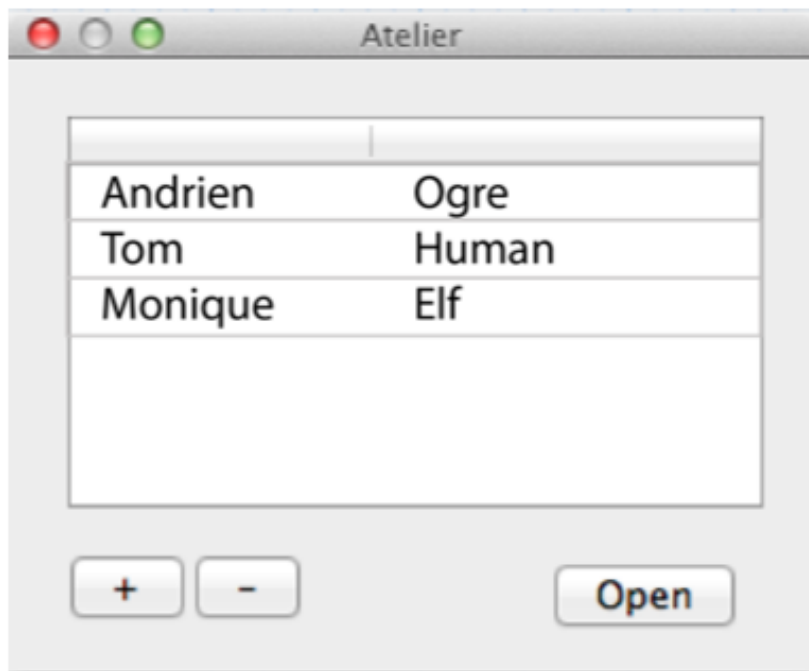
L'atelier : permet la conception de personnages pour notre jeu :

- (a) Définition de l'aspect visuel et des propriétés des personnages,
- (b) Définition des compétences, des comportements v.a.v. des autres personnages,
- (c) Définition des stratégies pour organiser leur mouvements et leur actions dans le labyrinthe.



Sujet du projet : un labyrinthe

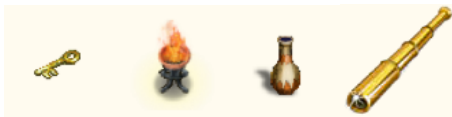
- **L'atelier**



Sujet du projet : un labyrinthe

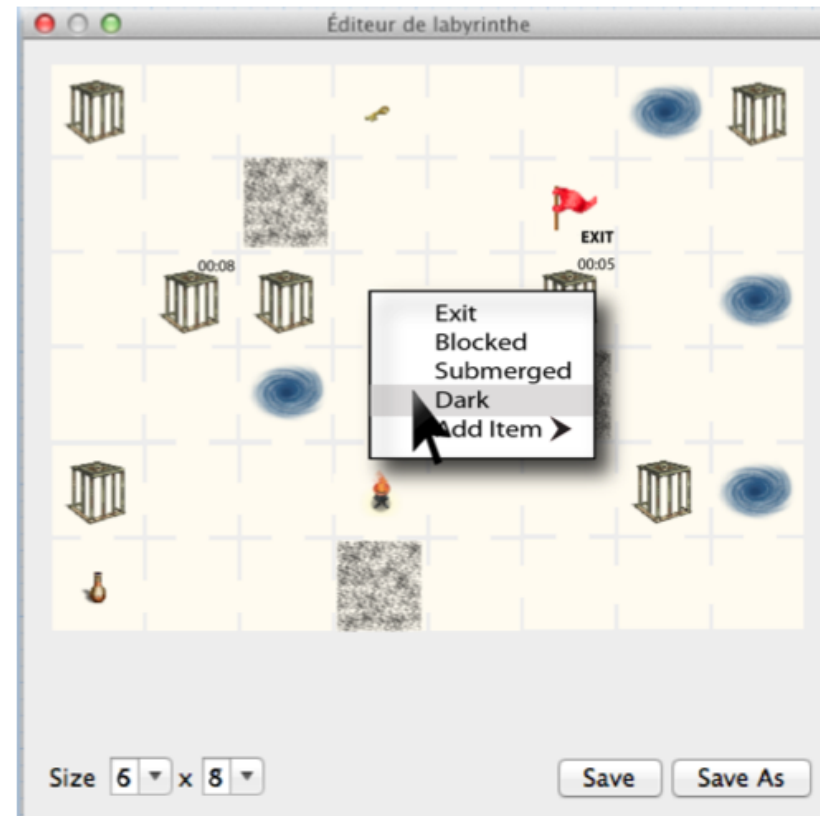
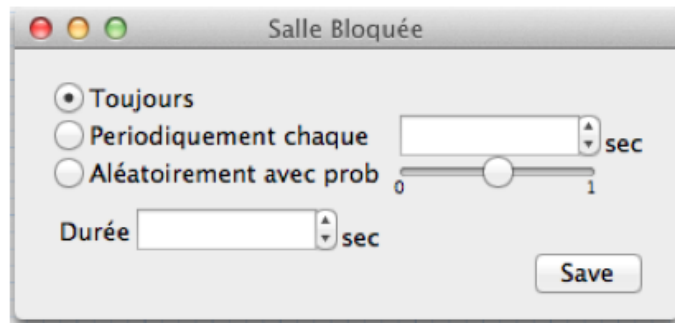
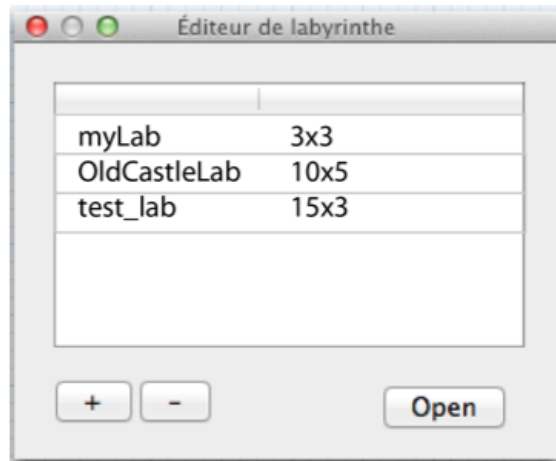
L' éditeur de labyrinthe : permet la conception d'un labyrinthe (grille à deux dimensions de salles) :

- (a) Choix de l'aspect visuel du labyrinthe, de la position des objets et des caractéristiques du terrain de chaque salle du labyrinthe.
- (b) Changement des propriétés de salles qui affectent la navigation des personnages.



Sujet du projet : un labyrinthe

- L'éditeur du labyrinthe



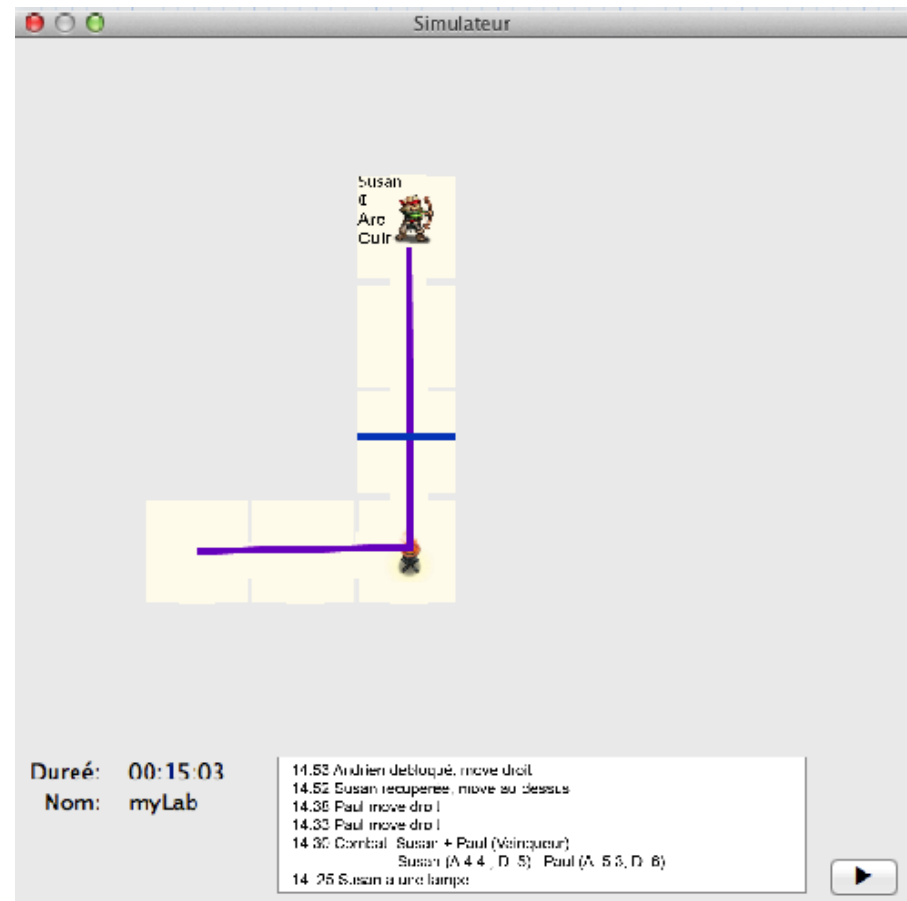
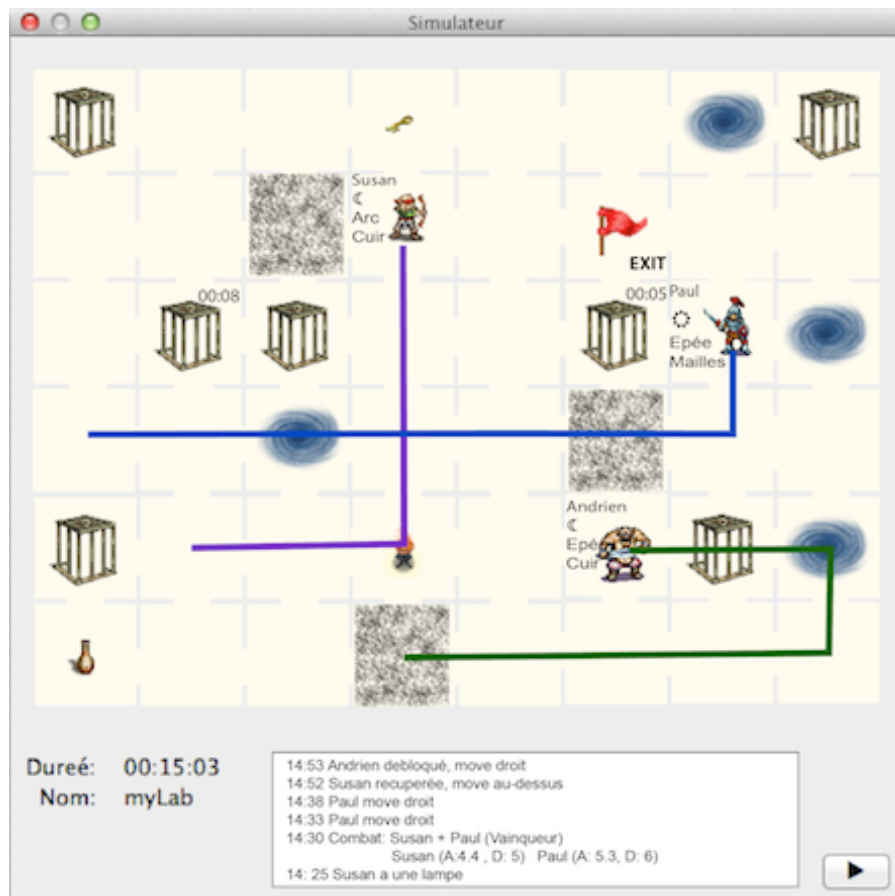
Sujet du projet : un labyrinthe

Le simulateur de jeu : simule l'avancement des personnages dans le labyrinthe jusqu'à leur sortie :

- (a) Simulation naïve où les salles et personnages ont des caractéristiques similaires, avec des stratégies de navigation simples,
- (b) Prise en compte des obstacles dans les salles spécifiées par le labyrinthe, le mouvement des personnages, ...
- (c) ... les compétences des personnages et des combats,
- (d) L'interaction directe sur le jeu (IHM) pour contrôler un personnage.

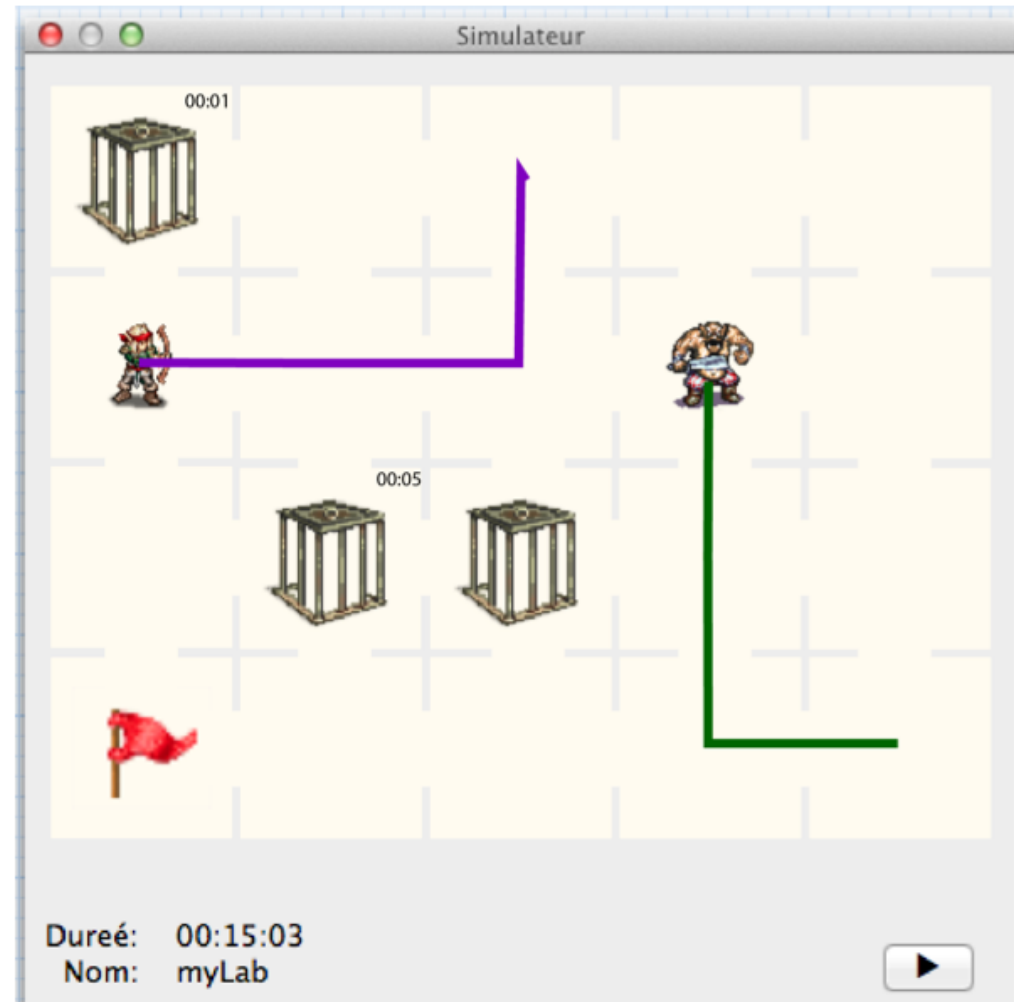
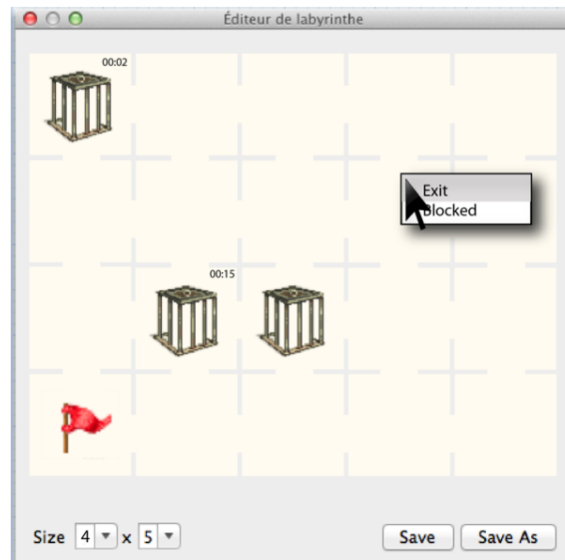
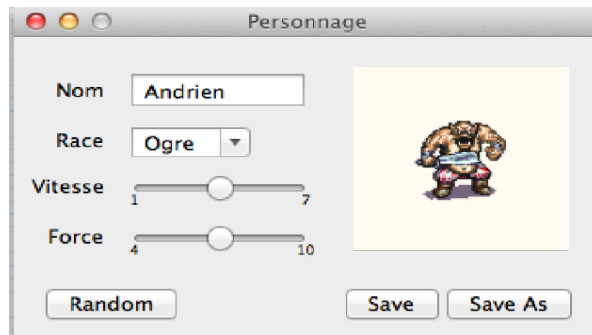
Sujet du projet : un labyrinthe

- **Simulateur**



Description détaillée : version 1 initiale

ce qui doit être fait à mi-parcours



version 1 - atelier initiale

ce qui doit être fait à mi-parcours

- **L'atelier**

- Permet pe l'utilisateur de :

- charger la description (propriétés) d'un personnage existant,
 - créer un nouveau personnage, éventuellement en modifiant les propriétés d'un personnage existant,
 - propriétés : son nom, sa race, sa vitesse de mouvement, sa force.

- Fonctionnalités souhaitées :

- Création, sauvegarde et chargement personnage,
 - Création par l'utilisateur, aléatoire, ou
 - en modifiant un personnage existant ("save as").

version 1 - éditeur initiale

ce qui doit être fait à mi-parcours

- **L'éditeur de labyrinthe**

- L'éditeur nous aide à définir un monde, qui est
 - une grille de taille NxM, avec des salles bloqués pour une période de temps (aléatoire, périodique, permanent) et avec une sortie;
 - avec une représentation visuelle de ces salles
- Nous aide à ouvrir et changer un labyrinthe existant
- Fonctionnalités souhaitées:
 - sauvegarde et changement du labyrinthe
 - placement des blocages (avec un temps de blocage)
 - placement des salles comme la sortie (unique dans le labyrinthe)
 - choix des images à mettre dans les salles
 - possibilité d'avoir un labyrinthe généré automatiquement

version 1 - simulateur initiale (p1)

ce qui doit être fait à mi-parcours

- **Simulateur** : permet à l'utilisateur de visualiser les positions des personnages et leur déplacements dans le labyrinthe.
- Le simulateur :
 - Contient une liste de tous les personnages,
 - Chaque personnage commence dans une salle,
 - Si une salle devient bloquée quand un personnage est déjà dedans (pendant la durée de son temps de mouvement), il doit attendre jusqu'au moment où la salle se débloque,
 - Un message est affiché (ex. console et log) chaque fois qu'un personnage arrive à la sortie.

version 1 - simulateur initiale (p2)

ce qui doit être fait à mi-parcours

- **Simulateur :**

Déplacement :

- Chaque personnage peut avancer vers une des 4 salles voisines de façon aléatoire, jusqu'à son arrivée à la sortie.
- Un personnage choisit la porte qu'il va chercher à ouvrir.
 - S'il arrive à ouvrir la porte (la salle n'était pas bloquée), il entre.
 - Si la salle est bloquée il essaye durant un certain temps (*temps de mouvement*) d'ouvrir la porte. À la fin de ce temps il peut essayer une autre salle ou éventuellement s'entêter à ouvrir cette porte.
 - Le temps de mouvement dépend de sa vitesse : choisissez une constante (*MaxMovementTime*) pour tous les personnages, et calculez leur temps de mouvement:

$$\text{Temps de Movement} = \text{MaxMovementTime} / \text{vitesse}$$

version 1 - simulateur initiale (p3)

ce qui doit être fait à mi-parcours

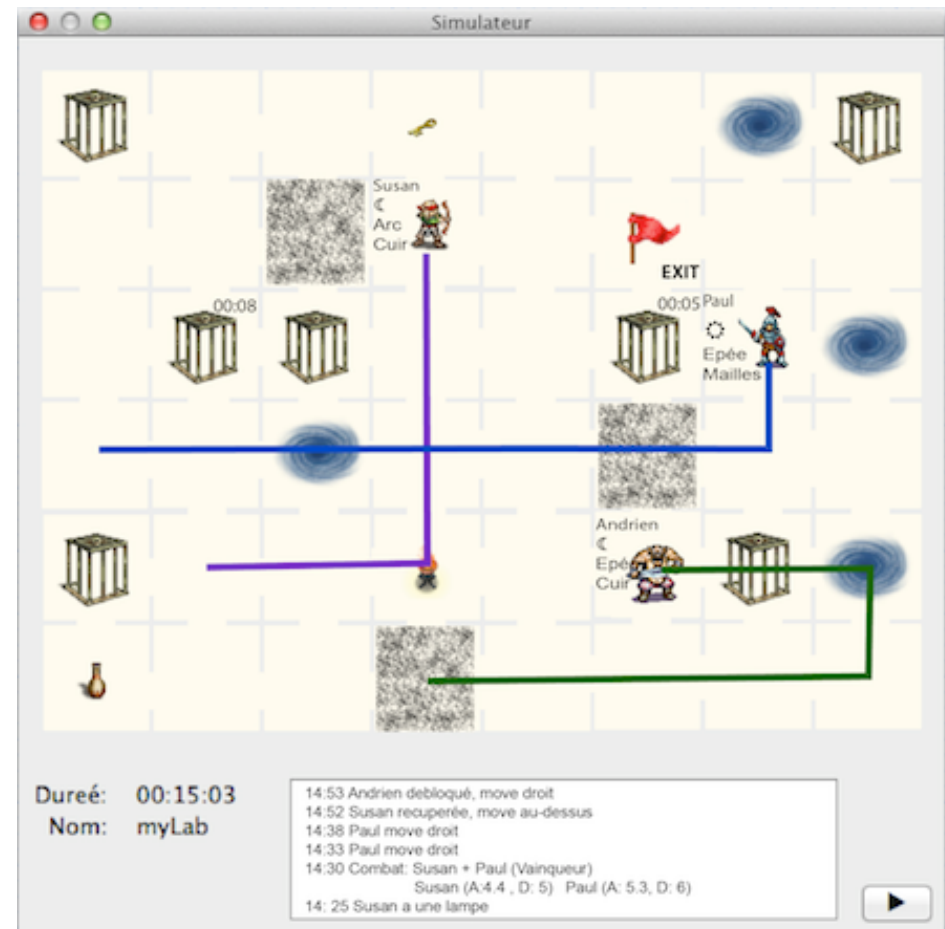
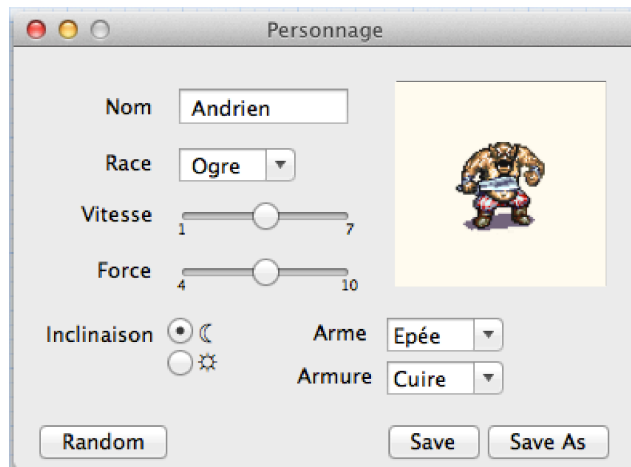
- **Simulateur :**

- **Fonctionnalités souhaitées :**

- Chargement de personnages créés et sauvegardés dans l'atelier,
 - Chargement d'un labyrinthe à partir du fichier créés dans l'éditeur,
 - Mettre des personnages dans les salles spécifiques ou aléatoires,
 - Visualisation du labyrinthe (salles et déplacements),
 - Sauvegarde dans un fichier de log les déplacements de tous les personnages et des événements importants (positions initiales, blocage, etc.). Cela aide trouver les erreurs de programmation plus facilement.

Description détaillée : version 2 avancée

ce qui doit être fait à la fin



version 2 – éditeur avancé (p1)

ce qui doit être fait à la fin

- **L'éditeur du labyrinthe avancé**
 - comme **version 1 +**
 - ajouter d'autres types des salles : noires (sans visibilité), submergés, ...
 - ajouter des objets dans des salles < clé, lampe de poche, jumelles, gilet de sauvetage, potion de soins >
- **Fonctionnalités souhaitées:**
 - **version 1 +**
 - sauvegarde et changement du labyrinthe avec les propriétés des salles et choix des salles (et images qui vont avec)
 - placement des objets pour aider avec les situations difficiles
 - possibilité d'avoir un labyrinthe généré automatiquement

version 2 – éditeur avancé (p2)

ce qui doit être fait à la fin

- **L'éditeur du labyrinthe avancé**
- Salles et Objets:
 - On a cinq types de salles : normales, bloquées, noires, submergées, et la sortie (unique)
 - Un personnage ne peut pas entrer dans une salle bloquée, sauf s'il a une clé. Là il peut traverser normalement.
- Un personnage peut traverser une salle noir ou une salle submergée, mais de façon plus lent (2 et 3 fois son temps de mouvement normal respectivement). Par contre, si le personnage a une lampe de poche il peut traverser les salles noires à sa vitesse normale. De même s'il a un gilet de sauvetage pour les salles submergées.

version 2 – atelier avancé (p1)

ce qui doit être fait à la fin

- **L'atelier avancé**

- comme **version 1** +
- ajouter une inclinaison,
- avoir des stratégies de mouvement plus élaborées,
- posséder un sac d'objets,
- l'équiper pour le combat (armure + arme)
- personnages contrôlés par l'utilisateur
- Fonctionnalités souhaitées :
 - Création, sauvegarde et chargement personnage.
 - Création par l'utilisateur, aléatoire, ou en modifiant un personnage existant ("save as").

version 2 – atelier avancé (p2)

ce qui doit être fait à la fin

- **L'atelier avancé**

- La stratégie de mouvement :

- Un personnage contrôlé par l'utilisateur n'a pas une stratégie de mouvement, l'utilisateur décide sa direction de déplacement
 - Un personnage autonome choisi de se déplacer vers une des 4 salles voisines de façon aléatoire. Si la salle est bloquée il doit attendre son temps de mouvement avant de ré-essayer (la même ou une autre salle).
 - Un personnage ne connaît pas le contenu des salles autour, sauf s'il a des jumelles dans son sac. Dans ce cas il peut avoir une stratégie plus avancée et décider quelle salle est la plus rapide à traverser et vides

version 2 – atelier avancé (p3)

ce qui doit être fait à la fin

- **L'atelier avancé**

- Rencontrer d'autres personnages:

Quand un personnage entre dans une nouvelle salle qui est occupée par un autre personnage il y a 4 possibilités :

- Si la salle est noire il progresse normalement comme ils ne peuvent pas se voir (mais de façon plus lente, sauf s'il a une lampe)
- Si le personnage qui occupe la salle est de la même race et inclinaison ils peuvent la partager et progresser
- Si le personnage est d'une autre race mais de la même inclinaison, il doit revenir en arrière dans la salle précédente ;
- Si le personnage est d'une autre inclinaison, il doit la combattre.

version 2 – atelier avancé (p4)

ce qui doit être fait à la fin

- **L'atelier avancé**

- Combat (1)

Chaque personnage qui entre en combat a un chance de succès, qui est une combinaison de son attaque, de sa défense, et de chance :

attaque = arme*force + chance (calculée pendant chaque combat),

défense = armure*vitesse + chance (calculée pendant chaque combat)

Armes et armures ont des coefficients différents.

- un arc peut avoir un coef de 1.1, une épée de 1.2, une hache de 1.3.
- une armure en un coef de 1.1, une armure de cotte de mailles de 1.3.

version 2 – atelier avancé (p5)

ce qui doit être fait à la fin

- **L'atelier avancé**

- Combat (2)

- Pour chaque personnage le simulateur calcule son attaque et sa défense, ainsi que 2 entiers aléatoires (chance) qui sont ajoutés aux attaques et aux défenses.

- Le résultat est

$\text{Resultat} = (\text{attaque1} - \text{defense2}) - (\text{attaque2} - \text{defense1})$.

- Si le Résultat est positif, le personnage 1 est le vainqueur, s'il est négatif, le personnage 2 est le vainqueur. Si le Résultat est 0 c'est un match nul et les deux personnages sont vainqueurs.

Un personnage vaincu doit rester dans la salle 3 fois son temps de mouvement (le temps de guérir), sauf s'il a une potion de soins qu'il peut boire et continuer normalement.

version 2 – atelier avancé (p6)

ce qui doit être fait à la fin

- **L'atelier avancé**

- Sac :

- Chaque personnage a un sac avec une capacité pré-défini.
 - S'il entre dans une salle avec un objet il doit l'ajouter dans son sac s'il y a de l'espace, la capacité du sac diminue alors d'une unité.
 - Un objet dans le sac peut être utilisé une seule fois, ce qui libère une unité de capacité dans le sac.
 - La seule façon de libérer de l'espace dans le sac est d'utiliser un de ses objet dans une salle correspondante.

version 2 – simulateur avancé (p1)

ce qui doit être fait à la fin

- **Le simulateur avancé** : permet à l'utilisateur de visualiser les positions des personnages avancés et leur déplacements, et les items dans le labyrinthe. De plus il permet à l'utilisateur de contrôler certains des personnages.
 - comme **version 1** +
 - calcule la quantité des objets restant dans les salles
 - calcule le résultats des combats entre deux personnages
 - s'il y a un combat dans une salle, les personnages qui essayent d'y entrer doivent attendre la fin du combat
 - si un personnage est contrôlé par l'utilisateur, le simulateur est dans la visualisation "fog of war"

version 2 – simulateur avancé (p2)

ce qui doit être fait à la fin

- **Le simulateur avancé**

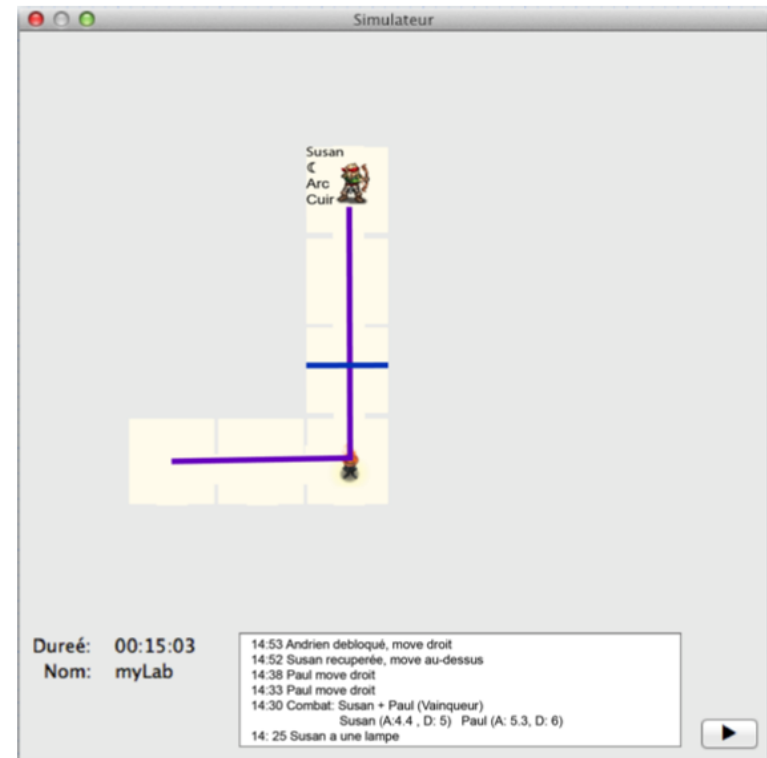
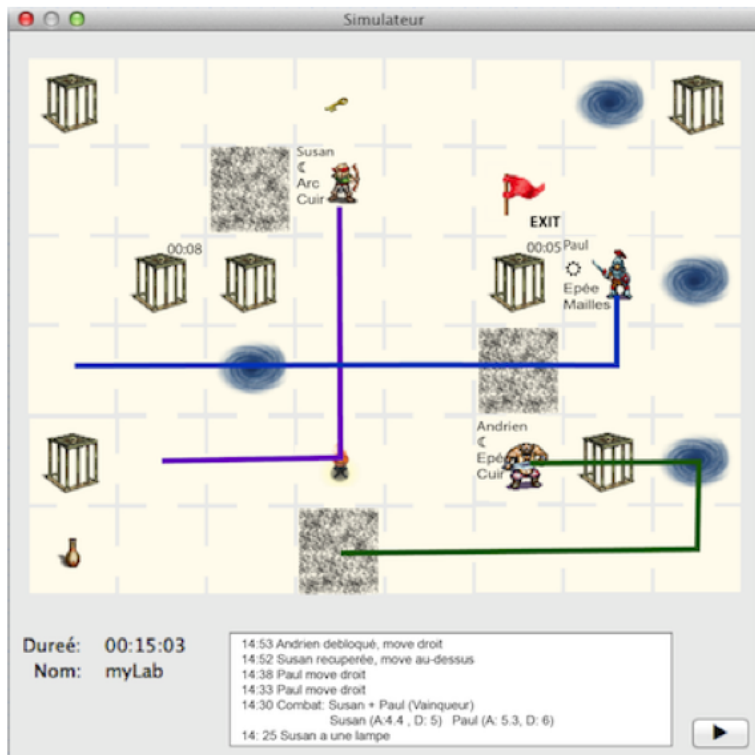
- Fog of war

- Si un personnage est contrôlé par l'utilisateur, le simulateur est dans la visualisation "fog of war"
 - C.a.d. la simulation tourne, mais les seules salles qui sont visibles sont les salles que le personnage de l'utilisateur a déjà traversé.
 - Exceptionnellement, si le personnage de l'utilisateur a des jumelles, il peut aussi voir les 4 salles adjacentes.
 - L'utilisateur peut donner des instructions de mouvement et de collection d'objets en utilisant le clavier ou la souris.

version 2 – simulateur avancé (p3)

ce qui doit être fait à la fin

- **Le simulateur avancé**
 - Fog of war (deactivé, activé)



version 2 – simulateur avancé (p4)

ce qui doit être fait à la fin

- **Le simulateur avancé**

- Fonctionnalités souhaitées

- Chargement de personnages créés et sauvegardés dans un fichiers au travers de l'atelier
 - Chargement d'un labyrinthe à partir du fichier crée dans l'éditeur
 - Visualisation du labyrinthe (salles et déplacements)
 - Sauvegarde de tous les événements dans un fichier de log
 - Gestions des combats
 - Gestions des quantités d'objets dans le labyrinthe;
 - Gérer les événements d'utilisateur pour contrôler un personnage
 - Visualisation particulière de la simulation quand il y a un personnage contrôlé par l'utilisateur ("fog of war").

Objectif du projet

- **Compétences à mettre en oeuvre**
 - sauvegarder/charger (personnages, labyrinthes)
 - construction d'une interface graphique
 - manipulation d'images, dessin
 - utilisation des threads (chaque personnage est indépendant) et synchronisation
- **Architecture et organisation des Classes**

Evaluation du projet

- **Evaluation**

- chaque sous partie est notée sur 13 (26/25 en totale)
- critères
 - fonctionnalités du code, propreté
 - mise en oeuvre des concepts vus en cours
 - qualité des **DEUX** soutenances
 - qualité du rapport et de la méthodologie (*cf. cours de GL*)
 - originalité (il faut aller plus loin ...)
 - réflexion sur la simulation (résultats vue les changements des paramètres)
- malus: en cas de retard, si le code ne fonctionne pas, si pas de rapport...

conseils

- Décomposez vos problèmes :
 - concentrez vous sur un sujet précis
 - plutôt que de tenter de réaliser les trois programmes en parallèle.
- Divisez votre travail entre vous
 - nous allons poser des questions sur votre partie
 - division naturelle ?
- **version release**
 - avoir toujours une version qui marche (même si pas complète)

conseils - test

- **test unitaire**

- tests qui vérifient la fonctionnalité d'une section spécifique du code
- tester les méthodes (surtout complexes) ou les classes
 - “corner cases”, branches dans le code
- automatiser ces tests

- **test d'intégration**

- tests qui vérifient les interfaces (connections/communication) entre les composants contre la conception/bût du logiciel
- tester d'abord en simplifiant des paramètres (ex. tester que un personnage, MovementTime très large ou petit, des labyrinthes très petites pour assurer des combats, des sacs infinis, predefinir les valeurs aléatoires pour les tests)
- plus difficile d'automatiser ces tests quand on gère des threads
 - tester d'abord avec 1-2 threads (personnages) et yield

conseils - code

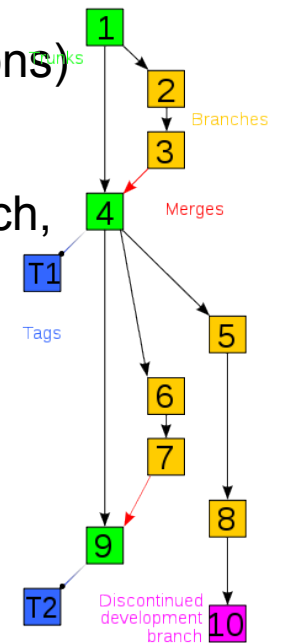
Pour gérer votre code (mais pas nécessaire) :

- **revision control (gestion de versions)**

- consiste à maintenir l'ensemble des versions d'un ou plusieurs fichiers (généralement [en texte](#)). Utilisée dans le domaine de la [création de logiciels](#), elle concerne surtout la gestion des [codes source](#)
- dépôt de copies locales (trunk ou branches, conflits des modifications)
- dépôt de copies distribués (pour collaboration et backup)
- notion de diff, checkout/clone/pull, add, commit/push, trunk et branch, conflict et merge

- **utils**

- svn, cvs, git, mercurial, etc
- e.g. de dépôt sur ligne/distribué : Bitbucket



fin de la présentation du projet
projet détaillé sur le web