# The Big Adventure

## Objectif général

Le but de ce projet de jouer à un jeu d'aventure et/ou de fabriquer les mondes du jeu d'aventure.
Il s'agit d'un meta-jeu, où l'on peut prendre autant de plaisir à jouer que de plaisir à créer les mondes du jeu. En terme d'expérience de jeu, on veut se rapprocher d'un mode de jeu à la zelda moderne (Breath of the Wild) pas dans son histoire/scénario, mais dans la capacité à créer sa propre histoire.

## Condition de rendu

Ce projet est à faire par binôme (2 personnes, ni 1 ni 3) et doit être déposé au plus tard le 19 janvier 2024 à 23h59 sur e-learning.

Une soutenance « bêta » aura lieu la semaine du 11 décembre (la date et l'horaire vous seront précisés). Votre travail ne sera pas noté, mais vous devrez tenir compte des remarques pour améliorer votre projet jusqu'au rendu final.

## Cahier des charges

Un niveau du jeu consiste en une carte (map) définie sous forme de grille (grid) contenant des obstacles et éléments décoratifs fixes ainsi que des éléments amis/ennemies, des éléments d'inventaires (dont font parties les éléments nutritifs).

1. **Les éléments décoratifs**
   Les ALGAE, CLOUD, FLOWER, FOLIAGE, GRASS, LADDER, LILY, PLANK, REED, ROAD, SPROUT, TILE, TRACK, VINE sont les élements décoratifs de la carte sur lesquels les personnages peuvent marcher.

2. **Les éléments obstacles**
   Les BED, BOG, BOMB, BRICK, CHAIR, CLIFF, DOOR, FENCE, FORT, GATE, HEDGE, HOUSE, HUSK, HUSKS, LOCK, MONITOR, PIANO, PILLAR, PIPE, ROCK, RUBBLE, SHELL, SIGN, SPIKE, STATUE, STUMP, TABLE, TOWER, TREE, TREES, WALL sont les élements qui peuvent servir d'obstacle pour éviter qu'un joueur accède à une partie de la carte.

3. **Les éléments intermittents**
   Les BUBBLE ou DUST apparaissent lorsque l'on rentre dans une zone déterminée de la carte pour attirer le regard du joueur sur un endroit précis de la carte.

4. **Les éléments amis ou ennemis**
   Les BABA, BADBAD, BAT, BEE, BIRD, BUG, BUNNY, CAT, CRAB, DOG, FISH, FOFO, FROG, GHOST, IT, JELLY, JIJI, KEKE, LIZARD, ME, MONSTER, ROBOT, SNAIL, SKULL, TEETH, TURTLE, WORM sont soit des amis, soit des ennemis. Ils peuvent parler avec un joueur, échanger des éléments de l'inventaire, voler des éléments de l'inventaire, faire des dégats au joueur et se transformer en fantôme.

5. **Les éléments d'inventaire**
   Les BOOK, BOLT, BOX, CASH, CLOCK, COG, CRYSTAL, CUP, DRUM, FLAG, GEM, GUITAR, HIHAT, KEY, LAMP, LEAF, MIRROR, MOON, ORB, PANTS, PAPER, PLANET, RING, ROSE, SAX, SCISSORS, SEED, SHIRT, SHOVEL, STAR, STICK, SUN, SWORD, TRUMPET, VASE sont les éléments de l'inventaire. Ils peuvent être échangés avec les amis.

6. **Les éléments nutritifs**
   Les BANANA, BOBA, BOTTLE, BURGER, CAKE, CHEESE, DONUT, DRINK, EGG, FRUIT, FUNGUS, FUNGI, LOVE, PIZZA, POTATO, PUMPKIN, TURNIP peuvent être mangés pour regagner de la santé. De plus les, BUNNY, CRAB, FISH, FROG et SNAIL peuvent être mangés s'ils sont cuits.

7. **Personnages**
   Ce sont les personnages que le joueur peut choisir en début de jeu : BABA, BADBAD, FOFO, IT. Il n'y a pas de différence entre eux, ce sont juste différents "skins".

8. **Les éléments de transports**
   PLANE/ROCKET/UFO (CLOUD), CAR (ROAD), TRAIN + CART (TRACK), BOAT (WATER) sont des éléments sur lequel le joueur peut monter pour être transporté d'un endroit de la carte à l'autre. Les PLANES/ROCKET/UFO, CAR et TRAIN se déplacent en suivant respectivement les CLOUD, ROAD et TRACK. Si le joueur se place devant un élément de transport, il se fait écraser. Si un joueur attend en haut ou en bas si la pîste est horizontale (et à gauche ou à droite si la piste est verticale), le moyen de transport s'arrête devant le joueur et le joueur peut monter dessus (dans le cas du train, on ne peut monter que sur les wagons). Un PLANE/ROCKET/UFO se déplace 2 fois plus vite qu'un joueur, un TRAIN se déplace 1.75 plus vite qu'un joueur et un CAR se déplace 1.5 plus vite qu'un joueur, un BOAT se déplace aussi vite qu'un joueur. Pour sortir du PLANE/ROCKET/UFO, TRAIN, CAR, BOAT le joueur utilise les flèches de déplacement. Par défaut, le PLANE/ROCKET/UFO, TRAIN, CAR suit sa piste, arrivé en bout de piste (si la piste n'est pas circulaire), il fait demi-tour (à vous de gérer correctement le demi-tour du TRAIN). Un BOAT reste à son emplacement initial et c'est le joueur qui le déplace.

9. **Les éléments d'armes**
   Il existe plusieurs armes. Une fois récupérés soit en passant dessus, soit en l'échangeant, une arme permet de causer des dégats sur les ennemies (elle ne fait rien sur les amis, c'est un jeu pour enfant !).
   - STICK, SHOVEL, SWORD: inflige des dégats aux ennemis qui se trouvent devant le joueur (en fonction de la direction du personnage). Le STICK et SWORD peut être enflammé s'il touche le feu (FIRE, LAVA). Dans ce cas, il reste enflammé tant qu'il ne touche pas de l'eau (WATER). Un STICK/SWORD enflammé permet de faire plus de dégat, de faire fondre la glace (ICE) et de cuire les éléments que l'on peut cuire (pour les manger). Une épée (SWORD) permet aussi de couper du bois (si elle n'est pas enflammé), en transformant les TREE/TREES en BOX. Une pelle (SHOVEL) peut aussi planter une graine (SEED) dans l'herbe (GRASS).
   - BOLT(arme): lancer des éclairs (qui se propage en fonction de la direction du personnage), tues les ennemis, met le feu aux arbres

10. **Les élements pour passer d'une map à l'autre**
    Les DOOR, GATE, HOUSE, TOWER si elles sont ouvertes (avec des KEY, LEVER) représentent des passages pour aller d'une map à une autre. Par exemple, pour qu'un joueur puisse aller dans une maison pour rencontrer un ami.

11. **Les éléments de biome**
    Les éléments ICE, LAVA, WATER sont des éléments qui sont hostiles au joueur, celui-ci ne peut pas se déplacer dessus. Mais ICE peut être fondu par le feu, et WATER peut être traversé en utilisant soit une BOX, soit un BOAT. Une BOX dérive en fonction du courant tandis qu'un BOAT est contrôlé par le joueur (s'il est dessus).

12. **Les éléments spéciaux**
    - BOOK, PAPER: explique le lore/folklore de votre jeu
    - BOX: flotte sur l'eau, est créé à partir d'un arbre, bouge avec le courant
    - BUCKET: prendre de l'eau, éteindre le feu, pour faire pousser un SPROUT
    - FIRE: brule les éléments, se propage doucement, cuit les BUNNY|CRAB|FISH|FROG|SNAIL
    - GHOST: passe à travers les murs (BED, BRICK, DOOR, FENCE, GATE, PILLAR, PIPE, STATUE, TABLE, WALL)
    - KEY, LEVER: ouvre les portes, maisons, gates, locks, tours
    - MIRROR: transforme les personnages fantômes en vrai personnage
    - SEED: planter avec une SHOVEL, se transform en SPROUT
    - SPROUT: se transform en arbre (TREE) si arroser avec un BUCKET plein d'eau.
    - WIND: pousse le personnage dans la direction du vent (avec la propriété flow)

# Les cartes

Les cartes sont des grilles sur lesquelles le joueur peut se déplacer. Une carte est définie pas un fichier ".map" qui contient une description textuelle d'une carte. Le format d'une carte contient plusieurs sections décrivant, la grille de la carte,

- **La grille de la carte**
  La grille est définie par une section `[grid]` qui définie
  - `size`: la taille de la grille width x height. Par exemple,

    ```
    size: (6 x 5)
    ```

  - `encodings`: l'encodage de chaque élément sous forme nom de l'élement puis l'encodage de l'élément sous forme d'un seul caractère entre parenthèse. Par exemple,

    ```
    encodings: WALL(W) BRICK(B)
               FENCE(F)
    ```

  - `data`: un text commençant par """ se finissant par """ utilisant l'encodage défini dans `encodings`. Par exemple,

    ```
    data: """
    WWWWWW
    W    W
    W FF W
    B    B
    BB BBB
    """
    ```

    Comme en Java/Python, les espaces à gauche du deuxième """ ne doivent pas être pris en compte.

- **Les éléments de la carte**

Les éléments de la carte sont définie par la section `[element]` qui définie des sous-sections `[element]` qui définissent les éléments. Par exemple, le joueur est défine comme ceci

```
[element]
  name: John
  player: true
  position: (1,1)
  health: 10
```

une épée qui traine et qui peut être ramassée par un joueur est définie comme cela,

```
[element]
  name: Durendale
  skin: SWORD
  position: (3,1)
  kind: item
  damage: 15
```

le courant d'une rivière est défini comme cela,

```
[element]
  skin: WATER
  zone: (10, 10) (5 x 5)
  flow: NORTH
```

et un ennemi est défini comme cela

```
[element]
  name: Waldo
  skin: CRAB
  position: (1, 3)
  kind: enemy
  health: 10
  zone: (1, 1) (5 x 3)
  behavior: agressive
  damage: 10
```

Voilà les attributs possibles

- `name`: nom de l'élément (optionnel)
- `skin`: apparence (BOOK, STICK, BUNNY, etc)
- `player`: true | false, il devrait y avoir qu'un seul player
- `position`: position de départ sur la carte, le 0, 0 est le coin en haut à gauche
- `health`: point de vie, un entier positif
- `kind`: type d'élément parmi friend, enemy, item, obstacle
- `zone`: zone définie par une coordonnée et une longueur et une hauteur dans laquelle se balade l'élément
- `behavior`: comportement de l'élément parmi shy, stroll et agressive
- `damage`: dégat infligé à chaque attaque, un entier positif
- `text`: un text associé au même format que data dans grid, pour les BOOK ou ce que dise les personnages
- `steal`: une liste de skin d'élément que sont volés de l'inventaire s'il y a contact avec le joueur

- `trade`: une liste de pair skin et nom d'un élément (skin -> skin name) qui sont échangeables.
  Par exemple,

  ```
  trade: CASH -> KEY red, CASH -> PIZZA
  ```

  qui veut dire que l'on peut échanger du cash contre la clé rouge (qui doit être définie comme élément sans position) ou du cash pour une pizza.
- `locked`: indique comment débloquer l'élément en indiquant un KEY|LEVER et son nom, par exemple

  ```
  locked: KEY red
  ```

- `flow`: indique le sens du courant pour les éléments ayant la bonne skin dans la zone, les valeurs possibles sont NORTH, SOUTH, EAST and WEST. Si on veut un flow, NORTH EAST, on peut définir deux éléments sur la même zone.
- `phantomized`: true | false, transforme le personnage qui entre en contact en GHOST.
- `teleport`: nom du fichier .map si lorsque l'on passe à travers certain DOOR, GATE, HOUSE ou TOWER, on se téléporte sur une autre carte. De plus, il existe une valeur spéciale "BACK" qui indique que l'on se téléporte sur la map dont l'on venait (pour revenir après avoir visité par exemple une maison).

# L'esprit du jeu

On souhaite créer un jeu où

- On n'explique pas, on montre
  https://en.wikipedia.org/wiki/Show%2C_don't_tell
  Par exemple, moins vous avez de texte mieux c'est !
- C'est un open world, on doit pas se sentir enfermer par des murs ...
- Le jeu doit être immersif : l'interface graphique ne montre que la carte, pas de minimap, pas d'UI en plus
- Les personnages du jeu ont un nom affiché en dessous d'eux.
  le fait que Jean Claude se fasse tuer a plus d'impact que si c'est juste un crabe.
- Le joueur ne doit pas être bloqué, l'objectif à atteindre doit toujours rester clair.
  Vous pouvez utiliser le Foreshadowing et les éléments intermittents pour vous aider.
- L'action faite par la barre d'espace doit être évident pour le joueur.
- Le joueur doit combiner des choses pour atteindre le but. Il doit se sentir intelligent.
  Aller regarder cette video: Comment Zelda a révolutionné le jeu vidéo en openworld ?. Je veux la même chose :) mais en 2D.

# Les différentes phases de développement

Le projet a quatre phases de développement : phase0, phase1, phase2 et phase3.

1. `phase0`: c'est la phase que vous devez finir pour ne pas avoir 0. Le jeu doit être capable d'afficher une map avec des éléments décoratifs, des obstacles, des ennemis, une épée (donc pas de vrai inventaire). L'affichage n'a pas besoin de scroller avec le joueur. Le système de point de vie doit fonctionner (sans la notion de nourriture). Les ennemis ont juste le comportement stroll.
2. `phase1`: on doit avoir des amis avec qui on peut parler et échanger des objets, donc l'inventaire doit fonctionner. La notion de portes et de clés doit fonctionner. Le système de point de vie doit ajouter la notion de nourriture. La carte doit être centrée sur le joueur et scroller lorsque l'on se déplace (sauf dans les coins). Les armes STICK et SWORD doivent fonctionner.
3. `phase2`: on doit avoir le feu qui se propage, les trains/voitures/avions/bateau/boite qui fonctionnent, le courant de l'eau qui marche. On doit de plus être capable de planter des choses et de les faire grandir. Les armes SHOVEL, BOLT et stick/sword enflammés doivent fonctionner.
4. `phase3`: les mécaniques restantes où celles que vous voulez ajouter si jamais la mécanique a été validé par un enseignant sur le forum discord du projet. Toute amélioration non validée va être considérée comme un non-respect du cahier des charges et le projet sera noté zéro.

Attention, en termes de notation si tous les mécanismes d'une phase ne sont pas implantés, nous ne tiendrons pas compte des mécanismes des phases supérieurs. Vous devez implanter TOUS les mécanismes d'une phase avant de passer à la suivante. Et oui, cela veut dire que vous aurez zéro par exemple, si les portes marchent avec les clés, mais on peut passer à travers les obstacles.

# Le programme

- **Commande**
  Il est possible de paramétrer le jeu en ligne de commande à l'aide des options suivantes :
  - --level name.map: nom du fichier contenant la map à afficher (obligatoire);
  - --validate: permet de valider qu'une carte est correcte sans jouer au jeu
  - --dry-run: permet de se balader sur la carte sans que les ennemis/amis ne bougent
  - --add-elements name.map: permet d'ajouter des comportements supplémentaires (dans ce cas, la partie grid de la map est ignorée), si on a deux valeurs possibles pour une propriété d'un élément, la valeur de la map spécifiée est prioritaire sur la valeur du level.
- **Pour jouer**
  Le jeu se joue au clavier, avec les touches de direction (haut, bas, gauche, droite) pour se diriger. La barre d'espace sert à faire l'action courante qui dépend de l'arme que vous avez en main et l'élément devant le personnage. La touche 'i' permet d'afficher l'inventaire. On utilise les touches de direction pour sélectionner ce que l'on veut dans l'inventaire. On valide avec espace.
- **Cartes fournies**
  Le jeu doit venir avec deux cartes prédéfinies, `demo.map` qui montre toutes les

mécaniques du jeu sur une petite carte et `adventure.map` qui est la vraie grande aventure du jeu.
- **Validation des cartes**
  Comme les cartes sont éditables facilement, il y a de bonnes chances que vous et vos utilisateurs fassiez des erreurs en écrivant la carte. Votre programme doit donc valider les cartes en indiquant toutes les erreurs (pas la première) AVANT de démarrer le jeu (et cela vaut aussi pour les cartes où l'on arrive par téléportation). Les messages d'erreurs doivent indiquer précisément le problème (avec un message en anglais) ainsi que la ligne ayant causé le problème. Attention, c'est une partie importante du projet, faite en sorte de bien gérér tous les cas d'erreur possible (sans que le code soit affreux SVP).
- **Affichage**
  Pour l'affichage, on vous demande d'utiliser la bibliothèque Zen 5.
  Attention à bien appeler la méthode `render` une seule fois par frame du jeu, pas plus sinon votre jeu va se transformer en sapin de noël (c'est la saison, c'est beau aussi, mais ça fait mal à la tête).
  Pour les tuiles, nous allons utiliser celle du jeu "Baba is You", https://babaiswiki.fandom.com/wiki/Category:Nouns.

# Références

Ant Manual pour le fichier `build.xml`.
How to create an executable jar ?.
JavaDoc.
Les entrées/sorties sur fichier.

# Conditions de rendu

Le projet est à rendre au plus tard le 19 janvier 2024. Le format de rendu est une archive au format zip (**tout rar, tar.gz, 7z et autre ne sera pas ouvert**) contenant :

- un répertoire `src` contenant les sources du projet ;
- un répertoire `maps` contenant les deux maps: demo.map et adventure.map ;
- un répertoire `docs` contenant le manuel de l'utilisateur (`user.pdf`) et le manuel qui explique votre architecture (`dev.pdf`) au format **PDF** avec une **section dédiée** aux améliorations/corrections apportées depuis la soutenance bêta ;
- un répertoire `classes` **vide** dans l'archive et qui contiendra les classes **une fois compilées** ;
- un répertoire `lib` contenant les libraries dont dépend l'application ;
- un jar exécutable `thebigadventure.jar` qui fonctionne avec `java -jar thebigadventure.jar` et donc qui possède un fichier manifest adéquat ;
- un fichier `build.xml` (écrit à la main) qui permet de :
  - compiler les sources (target compile) ;
  - créer le jar exécutable (target jar) ;
  - générer la javadoc dans `docs/api` (target javadoc) ;

    - nettoyer le projet pour qu'il ne reste plus que les éléments demandés (target clean).

  La target par défaut doit créer le jar.
  Pour vous aidez, il y a le manuel de Ant

Cette archive Zip (attention à l'encodage) aura comme nom `Nom1_Nom2_TheBigAdventure.zip`, où les noms sont ceux des membres du binôme par ordre alphabétique. L'extraction de cette archive devra créer un répertoire de nom `Nom1_Nom2_TheBigAdventure` contenant tous les éléments demandés ci-dessus.

# Notation

- Cas de 0 sans aucune correction (mort subite) :
    - projet **non effectué en binôme** (c'est-à-dire 2 personnes !) sans l'accord préalable de l'intervenant de TD ;
    - absence à la soutenance bêta ;
    - projet envoyé après la date ;
    - projet non déposé sur elearning ;
    - une archive qui n'a pas le bon nom ;
    - fichier d'archive dont l'extraction ne produit pas un répertoire qui a le bon nom ;
    - le projet utilise une ou des librairies externes autres que celles indiquées dans le sujet ;
    - présence de code copié / collé du net ;
    - l'utilisation de classes de `java.io` autre que `InputStream`/`OutputStream`, `Reader`/`Writer` et l'exception `IOException` ;
    - l'absence des fichiers user.pdf (doc utilisateur) et dev.pdf (doc de développement).
    - l'absence de javadoc, ou javadoc pas en anglais.
- Critères de notation :
    - Rémi Forax Junior (11 ans) doit pouvoir jouer à votre jeu ;
    - la propreté et la lisibilité du code auront un poids très important dans la note ;
    - l'architecture que vous aurez définie (interfaces, classes ...) devra être donnée dans les documents PDF et aura également un poids très important dans la note ;
    - pas de méthodes de plus de 20 lignes ! ;
    - pas de duplication de code, et respect des principes de programmation objet ;
    - pas de variable globale;
    - pas de code inutile ;
    - présence des différents rapports et, par conséquent, orthographe correcte !
    - prise en considération des remarques faites lors de la bêta pour le rendu final.

---