Projet C++

Introduction

Ce projet est un exercice dans lequel vous devez faire la démonstration que vous savez utiliser en c++ les éléments que nous avons vus ce semestre (la construction/destruction/copie, les exceptions, les énumérations, la redéfinition d'opérateurs, l'héritage, éventuellement la généricité, etc ...).

Mais **nous ne voulons pas vous voir utiliser des smartpointeurs ou des typage auto** car ils peuvent contourner les difficultés que nous traitons.

Naturellement votre programme devra être bien conçu, il ne suffit pas "qu'il marche". Idéalement vous pourrez illustrer quelques patterns que nous aurons vu ensemble.

Sur ce sujet les IA génératives semblent ne pas donner de solution globalement satisfaisantes. Vous pouvez vous aider de ce qu'elles vous proposent, mais il vous faudra tout retravailler plus finement et surtout être capable de défendre chaque partie de votre code à l'oral, à la fois en détail et dans la structure générale.

La soutenance sera longue, et donnera lieu à toutes sortes de questions.

Généralités

- Le projet est à faire en binôme. Les monômes ne seront pas acceptés sauf pour des questions de parité dans les groupes de TD ou des cas très exceptionnels. Un forum Moodle sera ouvert pour mettre en contact ceux qui cherchent un partenaire. Il n'y aura absolument pas de trinômes, et il est entendu que si des travaux se ressemblent trop nous prendrons les sanctions qui s'imposent.
- Vous déclarerez la constitution de votre binôme dans les 15 jours à venir en choisissant un groupe **binôme_xy** dans l'activité d'inscription mise en place sur moodle. Cela nous permettra de nous assurer que le travail a bien commencé, et d'organiser la suite.
- La soutenance aura lieu durant la période des examens, début janvier. Votre travail sera à rendre quelques jours auparavant. Nous vous donnerons les dates exactes lorsque les réservations seront confirmées.
- Attendez vous à ce que la note de soutenance soit individualisée. Chacun doit donc maîtriser l'ensemble du travail présenté, y compris si une partie n'a été développée que par votre camarade.

Présentation du jeu

"Baba Is You" est un jeu de réflexion qui se représente sur une grille bidimensionnelle dans laquelle un personnage se déplace pour atteindre un objectif. Il devra pour cela interagir avec les objets présents.

Sur la figure 1 vous êtes le petit lapin blanc (qui ressemble surtout à une patate), et il vous faut atteindre le drapeau. Sur la route, trois rochers vous font à priori obstacle. Il vous suffira d'en pousser un suffisamment pour vous ouvrir le passage. Tout cela est très classique.

Ce qui l'est moins ce sont les quatre phrases dans les coins, qui forment des règles qui expriment que :

- BA BA IS YOU baba est votre personnage
- FL AG IS WIN c'est en atteignant le drapeau que vous terminez ce niveau
- WA LL IS ST OP les murs sont des obstacles
- RO CK IS PU SH les rochers sont déplaçables

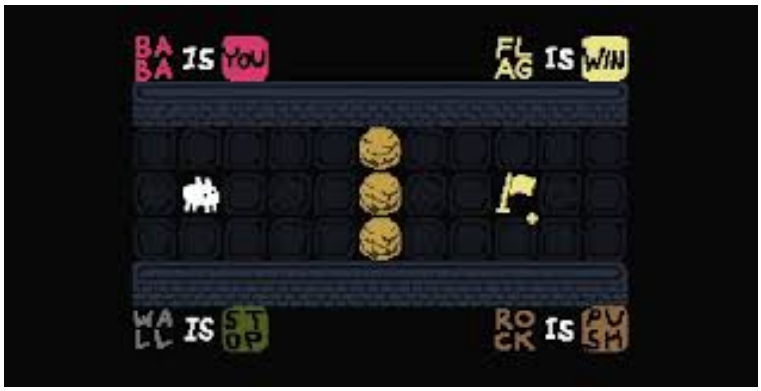

FIGURE 1 – Tutorial

Les membres gauche de cette règle correspondent à des objets présents sur la grille, et les membres droits sont des propriétés. Remarquez en particulier que ces mots occupent une case de la grille, et de fait il font partie du jeu : ces entités sont elles aussi déplaçables. En conséquence de quoi les règles peuvent évoluer : si la continuité des blocs de la phrase WALL IS STOP venait à être rompue, alors la règle ne s'appliquerait plus, et les entités pourraient traverser les murs. Dans la figure 1 baba n'a pas accès aux règles qui sont protégées par les murs, mais il le pourra dans d'autres niveaux. Vous pouvez vous familiariser avec le jeu sur ces démos :

- https://kbhgames.com/game/baba-is-you (limité à quelques niveaux)
- https://www.lexaloffle.com/bbs/?tid=142638 (pas très lisible : utiliser les flèches et X)

Nous nous limiterons à un petit nombre d'objets et nous ne vous demandons pas d'écrire des combinaisons de règles qui comporteraient le mot de liaison AND, ni de phrases complexes.

Cependant :

- une entité représentant un mot peut apparaître plusieurs fois : plusieurs règles peuvent donc la concerner. Vous pourrez établir une priorité entre propriétés pour arbitrer les résolutions.
- la lecture d'une règle peut se faire dans deux sens : de gauche à droite, et de haut en bas. Cela permet d'augmenter la jouabilité.

Méthodologie

Nous vous conseillons de vous faire une idée incrémentale des difficultés du sujet.

Comprenons nous bien : il ne s'agit pas de faire 4 versions, mais de vous assurer mentalement que votre modélisation intègre certains aspect avant de la retravailler pour en intégrer d'autres.

Voici un exemple de décomposition :

- En premier lieu : comprendre ce qu'il vous faut pour réaliser un jeu où l'on pousse des cubes. Dans cette phase vous réfléchirez à comment représenter un board très simplement, et comment en rendre compte visuellement. Vous pouvez par exemple privilégier un pattern Observateur/Observé entre la vue et le modèle. Si vous souhaitez effectuer une animation pour le déplacement, la vue d'une entité pourrait contenir des coordonnées du lieu de départ et du lieu de destination, ainsi qu'une pourcentage du chemin parcouru entre ces points, etc ... Faites vous une idée, mais n'allez pas au bout car d'autres difficultés ne sont pas prises en compte.
- La propriété d'être pushable ne concernera pas toutes les entités au même moment. Il y a aura des objets bloquants, des objets qui se feront traverser (donc des superpositions), et d'autres qui s'entredétruiront. A ce stade considérez les règles comme étant fixe : vous êtes toujours baba, les rochers sont toujours poussable, l'eau est toujours un élément de décors (superposable), les murs sont bloquants. Vous commencerez donc à décider de comment modéliser le fait qu'un objet peut potentiellement avoir une ou plusieurs propriété. Vous réfléchirez aussi à l'action de pousser en chaîne. Vous pouvez par exemple prendre les choses à l'envers : pour une poussée vers la droite, partez du bord droit et remontez vers la

gauche. Vous commencerez une accumulation dès lors que vous rencontrez une première case "libre" et ajouterez les éléments poussables que vous rencontrerez un par un. Si cette chaîne termine sur baba, alors vous saurez quoi déplacer.

Vous pouvez aussi aller en avant récursivement pour aller chercher une position libre après un bloc poussable. Trouvez votre propre solution, mais assurez vous toutefois que vous pourriez ainsi contrôler plusieurs babas de manière cohérente.

- Introduisez un système de règle, fixes mais définissable dans le code. Et parsez le à chaque tour pour vous assurer que vous pouvez fixer dynamiquement les propriété des entités. Testez par exemple en faisant en sorte qu'un tour sur deux une entité est push puis sera block au tour suivant, ou que vous pouvez changer qui est YOU . Déterminez les actions pour toutes les combinaisons plausibles de mots.
- Ajoutez un moyen de repérer les motifs des règles depuis l'état du board. Vous vous approcherez alors d'une version finale.

Quelques indications supplémentaires

Voici quelques exemples de figures de style (ou pattern) que nous aimerions voir dans votre code :

- La fonctionnalité `undo / redo` est une exigence naturelle pour permettre une bonne jouabilité. Elle vous obligera à avoir une réflexion sur la nature des copies à faire sur votre modèle, et sera l'occasion de faire des destructions soigneusement.
- Les propriétés sont décrites comme étant attachées aux entités, mais on souhaite aussi les récupérer de manière globale. Travaillez votre modélisation pour que les manipulations de ces propriétés soient confortables, naturelles, une fois que vous les aurez mises en place.
- Pour accéder plus agréablement qu'avec un getter à des informations, vous pouvez redéfinir l'opérateur `[]` sur votre plateau de jeu (appelons le `board`). Ainsi, par exemple `board[position]` vous permettra de récupérer toutes les entitées qui se superposent sur une position particulière ; `board[you]` ou `board[push]` pourrait être utilisé pour récupérer toutes les entités ayant la propriété indiquée.
- Le pattern Observateur / Observé a été évoqué un peu plus haut, entre le modèle et la vue, mais vous pouvez préférer une autre version du pattern Modèle / Vue / Controleur.

Si vous avez le temps

Voici une liste d'idée dans laquelle vous pouvez piocher pour individualiser un peu plus votre travail.

- introduire PULL qui serait déclenché par une combinaison de CTRL et d'une flèche de direction.
- ajouter KEY et DOOR
- gérer les règles de la forme WALL IS BABA
- soigner l'environnement du jeu (sauvegarde, suite de niveaux, éditeur)
- jouer sur le contrôle en permettant des règles comme A IS LEFT etc ... (c'est à dire reprendre en partie l'idée de : https://miknugget.itch.io/out-of-ctrl)
- traiter le AND

Conseils habituels

- Sauvegardez régulièrement votre travail. Chaque fois que vous envisagez une modification importante, conservez bien la version antérieure afin d'éviter des catastrophes. (Utilisez git par exemple)
- Si vous avez quelque part dans votre code un bloc qui fait plus d'une vingtaine de lignes, alors il est quasi-certain que vous devriez vous relire pour introduire une phase intermédiaire.
- Séparer le modèle et sa vue, c'est-à-dire ne mélangez pas les parties du code qui correspondent à ce qui est propre au jeu et celles qui correspondent à l'affichage.

- Illustrez ce que nous avons vu ensemble : ce qui est créé doit être détruit ; ne copiez que ce qui est nécessaire ; utilisez des références et pas seulement des pointeurs ; ajoutez des const où c'est le cas ; séparez bien votre code ; utilisez la stl à bon escient etc ...
- Surtout **n'utilisez pas** de smart pointers, ne typez pas vos objets avec "auto" : ce sont des choses qui vous permettent d'esquiver des difficultés sans que nous soyons sûr que vous compreniez.

Aspects pratiques de l'évaluation

Rendu

Les travaux seront à rendre sur `Moodle` sous la forme d'une archive **binomeNum.tar** (Utilisez impérativement ce format) [1] . Elle s'extraira dans un répertoire *`nom1-nom2`*, et contiendra :

- les sources et ressources (images ...) de votre programme ;
- un Makefile **compréhensible**
- un `README` qui indique succinctement ce qu'est ce projet, et comment effectuer la prise en main (compilation, exécution et utilisation). Le correcteur ne doit avoir que deux choses à faire [2] :
  1. décompresser votre dépôt dans une console unix ;
  2. lire votre fichier README et y trouver la ligne de commande qui lance le programme.
- un rapport au format *PDF* **rédigé** explicitant :
  - ce qui a été traité,
  - les aspects les plus significatifs,
  - les quelques problèmes connus,
  - l'état des extensions que vous n'auriez pas eu le temps d'implémenter.
  - un ou plusieurs diagrammes UML avec un niveau de détail suffisant pour comprendre votre architecture.

  Ce rapport ne doit pas être verbeux, ne cachez pas l'essentiel par un volume de baratin généré.
- toute chose utile pour rendre la soutenance fluide.

Soutenance

Elle se déroulera devant un ou deux enseignants dans un mélange de questions et de tests. Vous commencerez par une démonstration afin que nous ayons rapidement une vue d'ensemble ; nous étudierons votre architecture en nous basant sur votre diagramme ; vous nous guiderez dans votre code lorsque nous chercherons à vérifier la cohérence entre vos explications et votre réalisation. Il pourra vous être demandé d'écrire quelques lignes de code. Votre style sera scruté.

Vous pouvez préparer des supports pour répondre rapidement à des questions que vous anticiperiez. Dans tous les cas, il faudra privilégier les échanges avec le jury pour nous convaincre de vos acquis.

1. `tar cvf binomeNum.tar fichiers...` crée larchive mon_fichier.tar
2. Testez vous même votre rendu sur une autre machine