Serious Games
CodeEnBlocs

Rapport

Généralités

Date : 05/11/2016

Version : 3

Révision : 16/01/2017

Auteurs :

- Sofien Benharchache (Chef de projet)
- Thomas Bonnin
- Ali Chouenyib
- Alexandre Faucher
- Pascal Wagner

Résumé : jeu éducatif servant à l'apprentissage de l'algorithmique grâce à de la programmation par blocs, dédié aux élèves de 16 ans environ (niveau 3ème).

Type de jeu sérieux : Learning Game

Outils / technologies : C++, SFML

Introduction

Dans ce rapport, nous reprenons la dernière version du cahier des charges que nous avons rédigé et nous comparons les différences entre les prévisions du cahier des charges et le rendu du projet. Nous y avons également ajouté une section "Développement" située avant l'étude comparative.

Description du jeu

Contenu Sérieux

Public Principal : Collégiens 15 - 16 ans, niveau 3ème. *Lycéens 16 ans, niveau première.*

Contexte : Utilisation en classe avec encadrement, et pour les intéressés (certains niveaux sont complexes et demanderont beaucoup de réflexion, et représentent donc un grand défi, mais leur intérêt pédagogique est limité car le public est restreint). *Utilisation en classe avec encadrement. S'ancre dans le cours de Sciences de l'ingénieur pour le parcours première/terminale scientifique. Par rapport au cahier des charges préliminaire, nous avons revu l'âge du public principal pour qu'il s'inscrive mieux dans le contexte de l'apprentissage actuel au lycée.*

Objectif : Apprendre les bases de la programmation

Activités : Apprendre à réaliser de l'algorithmique de base sur les nombres entiers : additionner des nombres deux à deux, multiplier/diviser en utilisant des additions et des boucles, trouver le PGCD de deux nombres (grâce à l'algorithme d'Euclide), dire si un nombre est premier ou non (grâce au PGCD), etc...

Public Secondaire : Gens expérimentés à l'algorithmique. Tout âge confondu.

Contexte : Récréatif. Divertissement personnel.

Objectif : Améliorer ses bases en algorithmique, revenir sur des notions essentielles. S'amuser en programmant.

Activités : Identique au public principal, sans l'aspect éducatif ; il passera les différents tutoriels et se concentrera sur les challenges.

Gameplay

Le joueur a à sa disposition une série de blocs de données en entrée (nombres ou lettres), et des blocs algorithmiques. Grâce à ces blocs algorithmiques (opérations arithmétiques, instructions conditionnelles, boucles, etc…), il doit créer un programme qui va manipuler ces blocs d'entrées selon ce qui est attendu de lui en sortie (par exemple : faire l'addition de nombres deux par deux, multiplier deux nombres, ou encore, faire la moyenne d'une liste de nombres).

Au fil des niveaux, le joueur rencontre des énigmes de programmation de plus en plus difficiles, mais qui lui apportent de nouveaux blocs. Les blocs qu'il gagne au fil des niveaux lui permettent de résoudre les énigmes suivantes. La plupart du temps, l'obtention de ces blocs fait suite à un niveau où il devra obtenir le résultat de ce bloc, mais sans utiliser le bloc lui même : par exemple, il devra réaliser une multiplication de deux nombres avec les blocs d'addition (+), de condition (IF), et de boucle (WHILE); suite à la réussite de ce niveau, il obtiendra le bloc de multiplication (x), qu'il pourra utiliser par la suite. *Nous avons finalement supprimé les blocs conditionnels et les boucles, empêchant l'intégration de nombreux niveaux d'algorithmique, car nous nous sommes rapidement rendus compte que le moteur était plus long et complexe à réaliser que prévu (par exemple, faire correctement la liaison entre les blocs consécutifs et vérifier la syntaxe du code). Ainsi, nous avons choisi de nous concentrer sur le développement d'un moteur fonctionnel et ne permettant pas de bugs plutôt que sur l'ajout de blocs plus complexes pour le joueur.*

Objectif ludique :

Facteurs motivants majeurs :

- L'histoire (encore à compléter)
- Le challenge (trouver une solution, voire même la solution optimale)
- Encourage un apprentissage autonome

Le joueur peut s'amuser en codant. Grâce au système de gain de blocs, il peut revenir sur d'anciens algorithmes pour les améliorer, et accomplir des défis. La rejouabilité est limitée par le fait qu'une fois une notion acquise, il est moins intéressant de revenir dessus, le seul intérêt est d'optimiser le code si c'est possible.

Comment le gameplay permet-il d'atteindre l'objectif sérieux?

L'objectif sérieux est double : éducatif et applicatif.

Pour la composante éducative, à chaque niveau qui introduit un nouveau type de bloc (arithmétique, fonction, condition, etc…), le joueur suivra un petit tutoriel expliquant le fonctionnement de celui-ci. Par exemple, quand le joueur arrive aux niveaux utilisant des

blocs conditionnels (IF), on lui explique point par point ce qu'est une condition, une instruction, ce qui se passe lorsque la condition est valide ou non ; puis on lui montre un exemple. Ensuite, il doit réaliser un exercice simple utilisant la notion nouvellement acquise. *Nous n'avons pas intégré d'exemples de code, simplement une description textuelle du nouveau bloc et de son utilité.*

Pour la composante applicative, le joueur doit trouver une bonne manière d'utiliser les blocs à la manière d'un puzzle pour réussir le niveau, c'est-à-dire atteindre le résultat souhaité. Pour chaque énigme il y a une solution optimale (ex : nombre minimum de blocs utilisés, nombre minimum d'instructions exécutées. *Il ne s'agit que de seuil de nombre de blocs utilisés à atteindre.*) que le joueur peut chercher à atteindre (objectif secondaire). Pour faciliter l'apprentissage, le placement des blocs fonctionnera par drag-and-drop. *Le drag'n'drop a été supprimé au profit d'une méthode de navigation entre les lignes de code au clavier et un simple clic sur un bloc pour l'ajouter à la ligne de code actuelle.* De plus, des couleurs permettront de différencier les types de blocs par fonctionnalité (boucle, goto, test, calcul). Le jeu proposera également des défis du type "programmer le niveau sans utiliser tel ou tel bloc".

Structure du jeu

Le jeu sera composé d'une vingtaine de niveaux. *Finalement, seuls 8 niveaux ont été intégrés.* A chaque niveau, on donne au joueur les instructions qu'il devra réaliser ainsi qu'un exemple de sorties souhaité. Il peut prendre connaissance des entrées et des blocs de programmation à sa disposition pour réfléchir à une solution. Lorsqu'une sortie ne correspond pas à celle attendue, il lui est expliqué quelle sortie était attendue à la place. Lorsque toutes les sorties sont correctes, on lui indique le nombre de blocs écrits et le nombres d'instructions exécutés par son code, en comparaison à un nombre optimal (des challenges), afin que le joueur puisse établir par lui-même si son travail est suffisant ou s'il doit améliorer son code.

Un score lui est attribué sous la forme d'étoiles en fonction de sa distance au nombre de blocs optimum :

- *1 étoile : le niveau est réussi*
- *2 étoiles : le niveau est réussi en utilisant un nombre de blocs jugé plus optimal que la moyenne*
- *3 étoiles : le niveau est réussi en utilisant le nombre de blocs optimum*

Un arbre des niveaux est disponible en annexe.

Procédure d'évaluation

Pour rappel, ce jeu sera destiné à une utilisation en classe aux côtés d'un enseignant. Concernant l'évaluation, l'encadrant devra, à la suite d'une ou plusieurs sessions de jeu, proposer aux élèves un petit exercice à programmer cette fois en lignes de code, et plus en blocs, dans un langage haut niveau (ex : Python, Matlab).

Les puzzles du jeu servent également d'évaluation puisqu'il est nécessaire de comprendre le raisonnement derrière les algorithmes codés pour progresser et que le joueur doit réaliser les fonctions qu'il a lui-même codé dans les niveaux suivants.

En plus du puzzle lui même, la présence de challenges (nombre de blocs et nombre d'instructions exécutés optimales) constitue également une évaluation des performances du joueur.

Développement

Organisation

CodeEnBlocs a été développé en C++ à l'aide de la bibliothèque graphique SFML. Le projet a été développé from scratch, c'est pourquoi nous avons passé beaucoup de temps sur la création du moteur, côté modèle, et de tous les composants graphiques côté vue. En effet, pour le moteur, il a fallu créer notre propre langage et notre propre compilateur pour vérifier la syntaxe des blocs de code mis les uns à la suite des autres et les interpréter correctement. Du côté IHM également, nous avons dû créer toutes les interactions et les visuels en se basant sur SFML mais sans utiliser de surcouche.

Nous avons rapidement choisi de séparer le développement en deux parties : l'IHM et le moteur. Nous avons initialement travaillé à trois sur le moteur et à deux sur l'IHM. Cependant, lorsque nous avons commencé à vouloir intégrer le moteur à l'IHM, deux personnes ont dû se concentrer sur la liaison entre les deux parties. Cette stratégie a été particulièrement simple à mettre en place grâce aux outils GitHub (en particulier la création de branches), Slack et Trello.

Il est important de noter que nous souhaitions le plus possible suivre un Test Driven Development, c'est pourquoi nous avons mis en place des tests unitaires à peu près au même moment que nous avons commencé à développer le moteur. Ainsi, nous avons pu suivre l'évolution du comportement de notre production au fur et à mesure des modifications que nous apportions au code.

Design Patterns

Composite Pattern

Pour la définition des blocs de code au niveau des classes métier, nous avons choisi d'utiliser le Composite Pattern. En effet, les blocs prennent des arguments, et nous nous sommes aperçus que certains blocs pouvaient prendre d'autres blocs en argument, mais pas tous les types de blocs. Par exemple, on peut réaliser une addition (bloc +) entre une variable (bloc var) et un autre bloc addition. Ainsi, nous avons divisé les blocs en blocs composites, qui peuvent être appelés dans d'autres blocs, et les autres.

Factory Pattern

Le pattern Factory est utilisé dans l'interface graphique pour créer les blocs nécessaires pendant un niveau.

FSM Pattern (Finite State Machine)

Dans l'interface graphique, une machine à états est utilisée pour savoir à tout moment où on se situe dans le jeu (dans l'écran de sélection de niveaux, dans un niveau, etc.) et ainsi charger ce qu'il y a à charger au bon moment.

Command Pattern

Le pattern Command est l'un des pattern les plus important que l'on a du utiliser. Car c'est grace a ce pattern que nous pouvons avoir une méthode "execute()" dans chaque blocs. Cette méthode nous permet donc de commander l'exécution du bloc au moment de notre choix et d'avoir un comportement différent pour chaque bloc.

Singleton Pattern

Le pattern Singleton est utilisé pour associer un identifiant unique à chaque bloc lors de l'initialisation du programme.

Adapter Pattern

Le pattern Adapter nous a permi de fusionner facilement un Bloc IHM d'un Bloc de la partie métier.

Etude comparative

Cette étude comparative se fera sur les jeux/outils pédagogiques suivant :

- Human Resource Machine
- Hour of Code
- CodinGame
- Shenzhen I/O

Les critères de comparaison seront les suivants :

- Public visé
- Utilisé dans l'enseignement ? divertissement ?
- Courbe de difficulté
- Langage / Types de blocs choisi
- Types de puzzles (Déplacement d'un personnage, traitement de données ou de signal, etc…)

Autres Références :

- SpaceChem
- TIS-100
- Leek Wars
- Scratch (langage) (base SmallTalk) (utilisé dans l'enseignement : https://fr.wikipedia.org/wiki/Scratch_(langage))
- Stencyl (base Scratch)
- Prog and Play
- Untrusted (https://alexnisnevich.github.io/untrusted/)
- Apple Swift Playgrounds

Human Resource Machine

Interface du jeu Human Resource Machine

Human Resource Machine est un jeu vidéo dans lequel le joueur doit programmer les actions d'un employé afin qu'il effectue les tâches de son directeur. Pour cela, le joueur doit utiliser des blocs de programmation de très bas niveau (Jump, CopyFrom, Add, Inbox, etc…). A chaque niveau, le joueur doit réaliser une tâche différente (du simple copier-coller, à la multiplication et au modulo, en passant par l'addition de paires, d'une série nombres finissant par 0, etc…).

Une dizaine de niveaux proposés

La difficulté est l'optimisation. À chaque niveau est associé un nombre optimal de lignes écrites dans le code, et un nombre optimal de lignes exécutées. Il est souvent peu difficile de réussir la tâche, mais les challenges peuvent prendre parfois une heure entière par niveau.

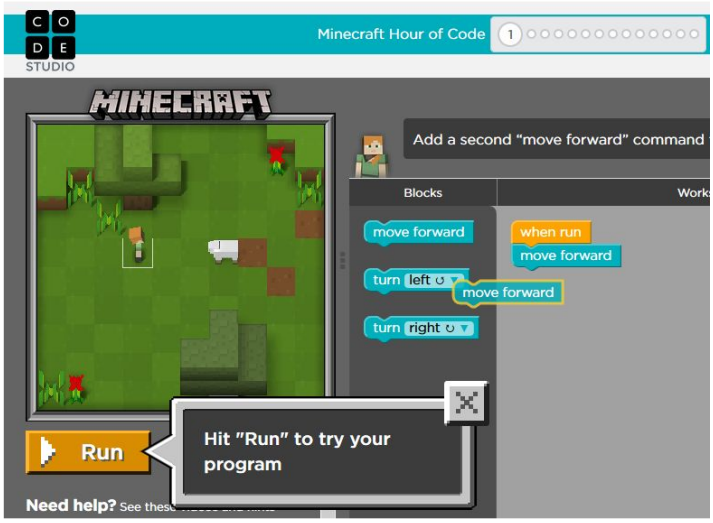
Exemple de challenge proposé

Human Resource Machine n'est pas un jeu sérieux mais nous pensons que cette idée des challenges comme objectifs secondaires pour optimiser son code peut être une bonne idée à intégrer dans notre jeu, car la simple notion de "défi" sera potentiellement une motivation chez une partie des joueurs, favorisant ainsi l'apprentissage.

Hour of Code

Code.org est un site web d'apprentissage de la programmation. L'organisation à l'origine du site œuvre pour rendre accessible à des professeurs, des étudiants ou toute autre personne intéressée par le sujet des outils et méthodes, dont des jeux vidéo, pour apprendre le plus efficacement possible la programmation.

Hour of Code est une catégorie d'exercices disponibles sur Code.org. Le concept est simple : il s'agit pour les professeurs de proposer à leurs élèves une suite d'exercices (généralement entre 10 et 15 niveaux) à réaliser pendant une heure. Il peut s'agir de jeux demandant de coder le déplacement d'un personnage comme d'un logiciel à réaliser pas à pas en étant guidé, le tout étant réalisé en programmation par blocs.



Interface des exercices sur le thème de Minecraft disponibles sur Hour of Code

Les blocs de code s'emboîtent à la manière de pièces d'un puzzle par drag'n'drop et respectent un code couleur. Ils sont de très haut niveau : aucun code n'est réellement affiché, les instructions sont représentées par des verbes d'action (ex : tourner, avancer, jouer un son, etc.). Certains blocs possèdent des paramètres, là encore il s'agit de rendre l'expérience utilisateur la plus simple possible : les paramètres sont des mots clairs à choisir dans une liste déroulante (ex : gauche / droite pour l'instruction tourner). Certaines séries d'exercices offrent toutefois la possibilité que le texte des blocs soit remplacés par la syntaxe d'un véritable langage de programmation comme JavaScript ou Python.

Le joueur a à sa disposition un certain nombre de blocs différents qui varie selon les niveaux et les jeux et il doit réussir le niveau avec une contrainte sur le nombre de blocs posés. Les niveaux sont constitués de deux parties : dans un premier temps, le joueur écrit son programme, puis dans un second temps il l'exécute et observe le déroulement de celui-ci pas à pas. Une fois le programme entièrement exécuté, le jeu affiche un feedback.

La particularité la plus notable de Hour of Code est que chaque jeu, ou série d'exercices, et assez court et indépendant des autres. Cela permet d'une part de proposer des notions ou langages différents et d'autre part d'affecter un thème propre à chaque jeu. Bien qu'il n'y ait pas d'âge maximum conseillé, la plupart du temps, ces thèmes visent un public jeune; on y retrouve par exemple La Reine des Neiges, Star Wars VII ou encore Minecraft.

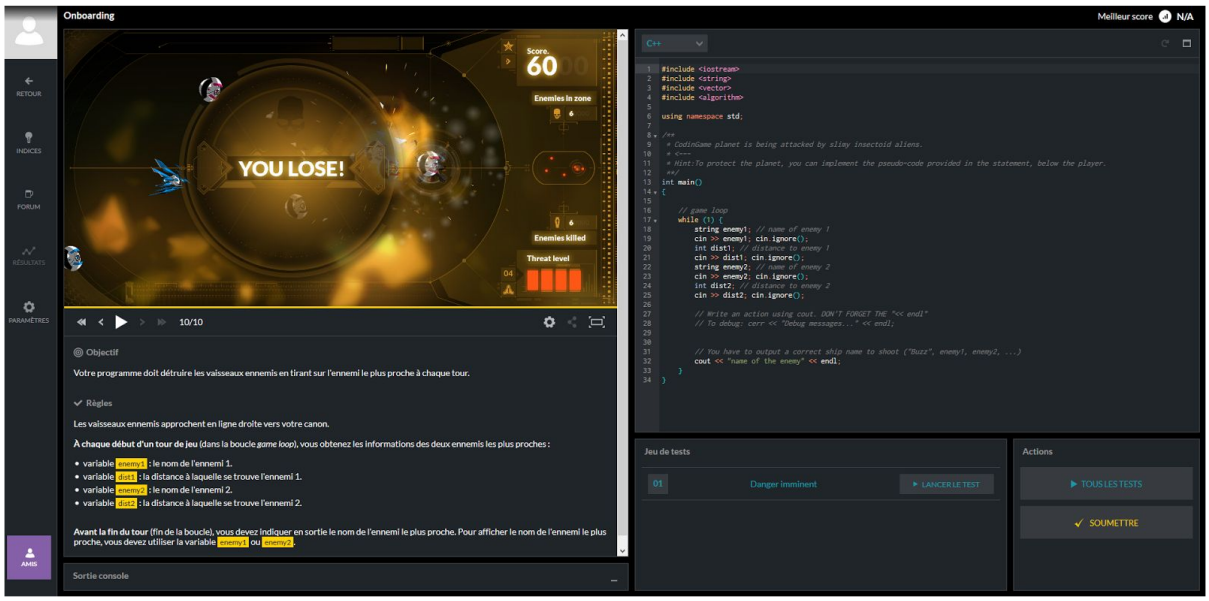
Hour of Code utilise comme nous la programmation par blocs pour l'apprentissage et vise globalement un public similaire à notre jeu. Bien qu'il ne propose jamais ses jeux à un public précis, on remarque grâce aux thèmes, à la difficulté faible des exercices et à l'aspect très haut niveau des blocs disponibles qu'il s'agit d'un public très jeune et surtout non initié à la programmation. Nous ciblons également ce type de joueurs, cependant, nous avons choisi de proposer des blocs de code plus bas niveau et de nous appuyer notamment sur les connaissances de mathématiques et de logique du joueur. Hour of Code, comme son nom l'indique, est conçu pour que les sessions de jeu durent une heure seulement. Notre jeu, lui, comportera plus de niveaux et la difficulté maximale sera bien supérieure à Hour of Code. L'idée est de proposer non pas dix exercices à réaliser en une heure mais une trentaine de niveaux dont les premiers pourront être réalisés en quelques minutes et les derniers en plus d'une heure, par exemple.

Les objectifs des niveaux sont également différents puisque dans notre jeu l'objectif de chaque niveau est de réaliser soi-même, à l'aide de blocs primitifs, une opération que l'on pourra réutiliser plus tard sous la forme d'un blocs : nous nous concentrons sur l'algorithmique. De plus, chaque bloc sera utilisable dans chaque niveau, contrairement à Hour of Code où les types de blocs sont restreints à chaque fois.

Finalement, Hour of Code pourra nous servir d'inspiration pour son excellente UI et UX : l'interface graphique et le gameplay sont particulièrement instinctifs et agréables à l'œil, ce qui est un point essentiel pour catalyser l'apprentissage. Il montre également que l'utilisation de blocs peut s'avérer vraiment efficace pour la compréhension de la programmation.

CodinGame

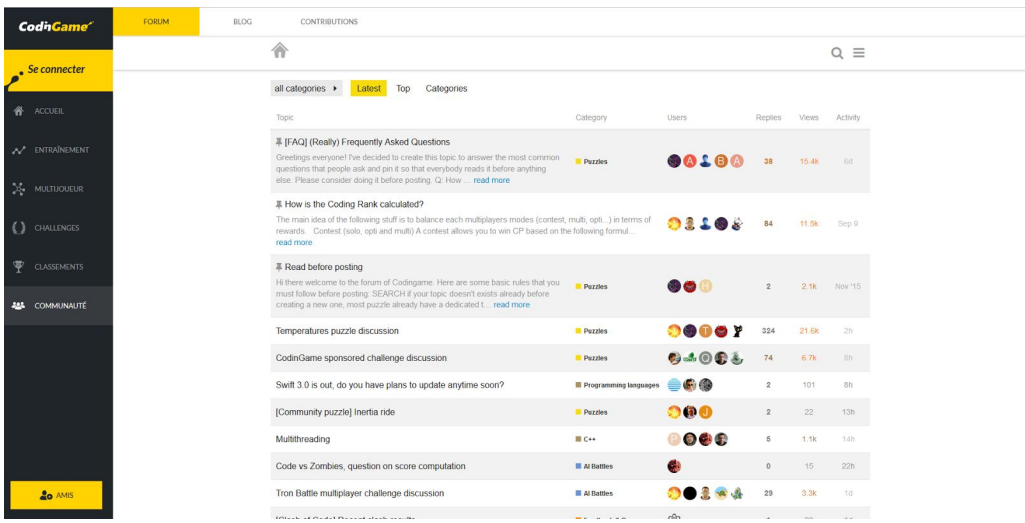
CodinGame est un site web proposant de nombreux jeux basés sur la programmation à destination des développeurs. Il s'agit plus d'un logiciel d'apprentissage et d'amélioration des compétences de programmation que d'un simple jeu. A chaque niveau, le joueur doit coder (dans le langage de son choix) une solution au problème posé, la forme de la solution pouvant changer du tout au tout d'un niveau à l'autre : du déplacement d'un personnage, à la gestion de robots de combats, en passant par une course de vaisseaux.



Premier niveau du didacticiel

Le challenge est constitués en deux composantes. D'abord, la satisfaction personnelle d'avoir complété un niveau (la même satisfaction que lorsqu'on code depuis plusieurs heures et que notre code fonctionne enfin). Ensuite, la compétitivité est également une part importante de Codingame ; des tournois sont souvents réalisés, et des prix sont distribués aux meilleurs.

Un dernier aspect apportant de la satisfaction au joueur est l'appartenance à une communauté de joueurs/développeurs, et de pouvoir s'entraider grâce au forum.



Forum de Codingame

Enfin, Codingame sert également de plateforme de recrutement. Les entreprises peuvent cibler bien plus efficacement les développeurs qu'ils veulent embaucher. Les joueurs peuvent donc se voir offrir un poste de développeur grâce à leurs compétences et ce en jouant.

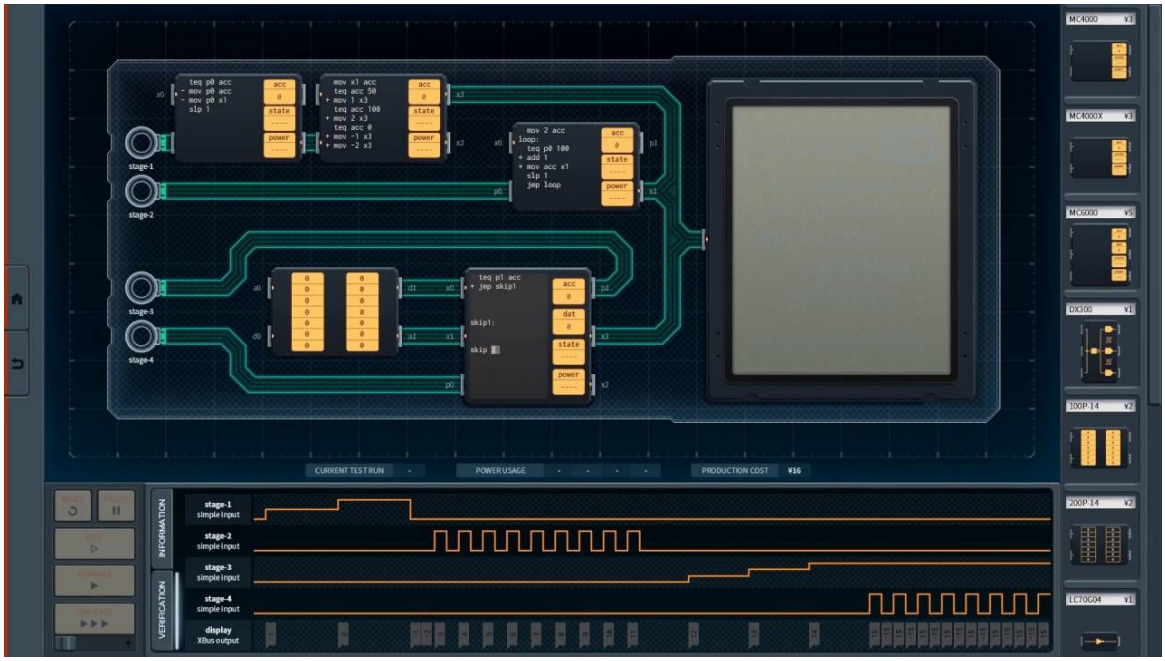
Topic	Users	Replies	Views	Activity
About the Jobs category		0	508	Sep '15
Competitive Programming vs Software Engineering		3	242	Sep 7
Need coders for my video game company		8	589	Jul 15
We're hiring our CodinGame Evangelist		35	7.5k	Apr 30
Sharing information on jobs & looking for work		7	731	Apr 10

There are no more Jobs topics.

Offres d'emplois disponibles sur CodinGame

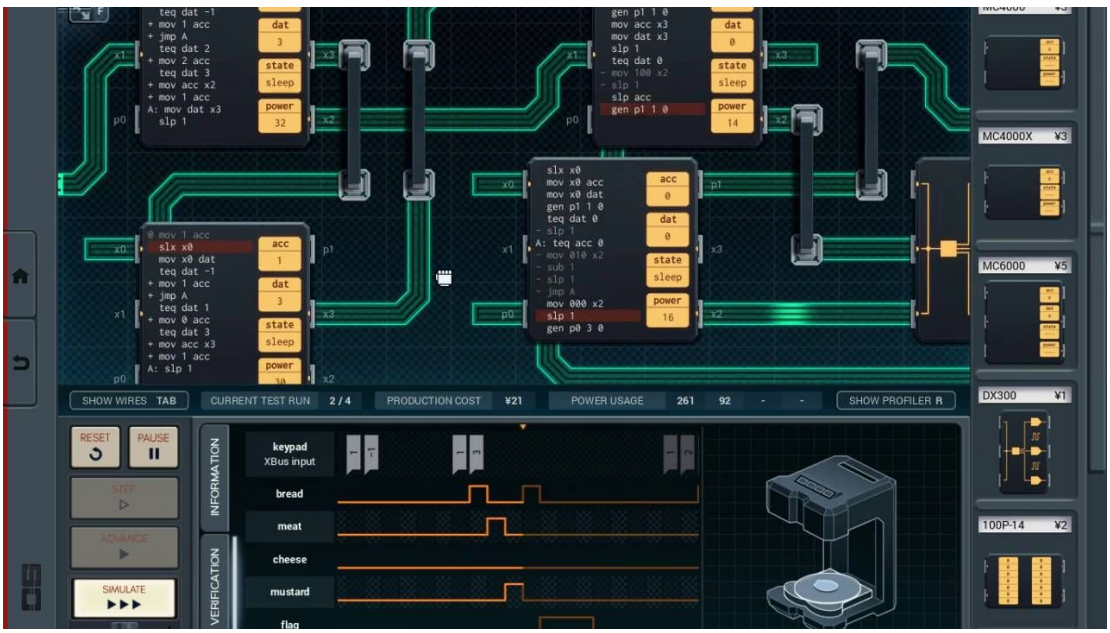
Bien que l'idée de se perfectionner et d'améliorer ses compétences est un élément clef qui guide notre level design global, a dimension sérieuse de CodinGame est donc bien différente de la nôtre et le public n'est absolument pas le même. Ici, les joueurs connaissent la programmation et utilisent même le langage qu'ils souhaitent pour progresser : on est loin de la programmation par blocs en drag'n'drop. De plus, nous ne pourrons pas apporter cette notion de communauté et d'émulation car notre jeu ne sera pas multijoueur.

Shenzhen I/O



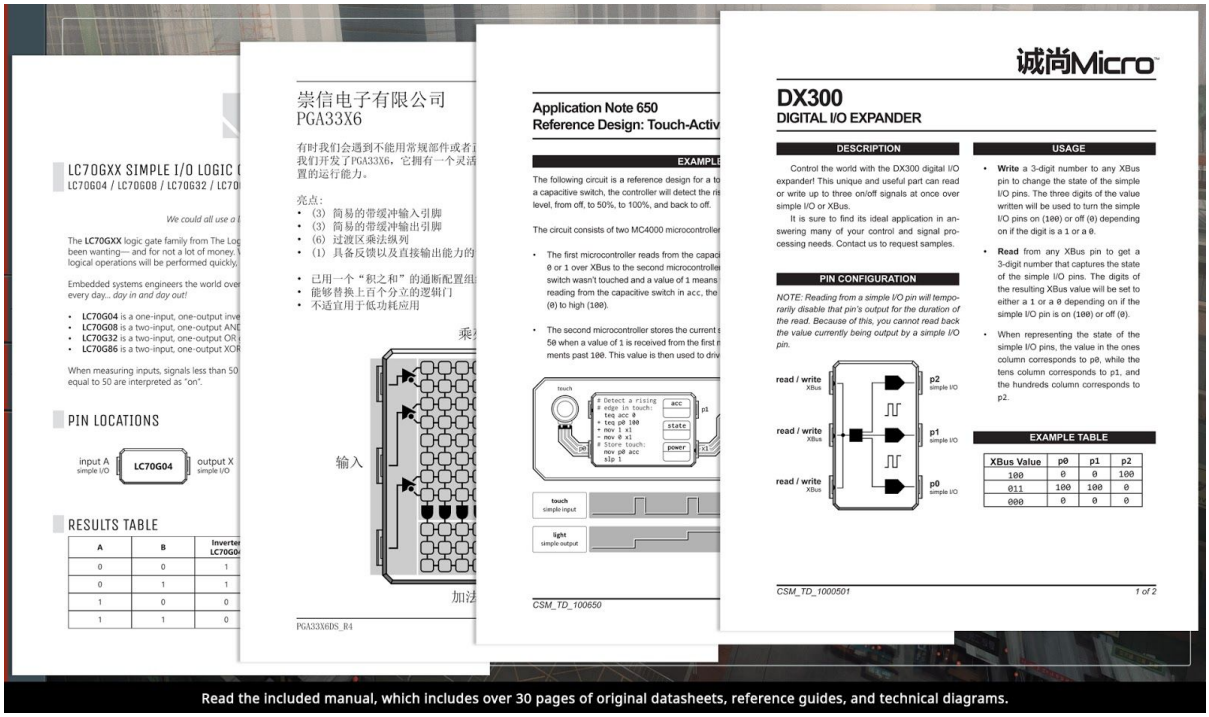
Interface du jeu Shenzhen I/O

Le joueur incarne un nouvel employé d'une société d'électronique à Shenzhen, ville caractéristique de la Chine par son essor économique depuis les annés 70. Le joueur est amené à réaliser des circuits imprimés et à programmer des puces électroniques afin de réaliser la tâche qui lui est demandée (de la simple fausse caméra qui imite une activité par des impulsions dans deux LEDs, à l'affichage de texte sur un panneau LCD à partir de signaux électriques, en passant par la réalisation d'un appareil à fabriquer des sandwich avec des options fromage et moutarde).



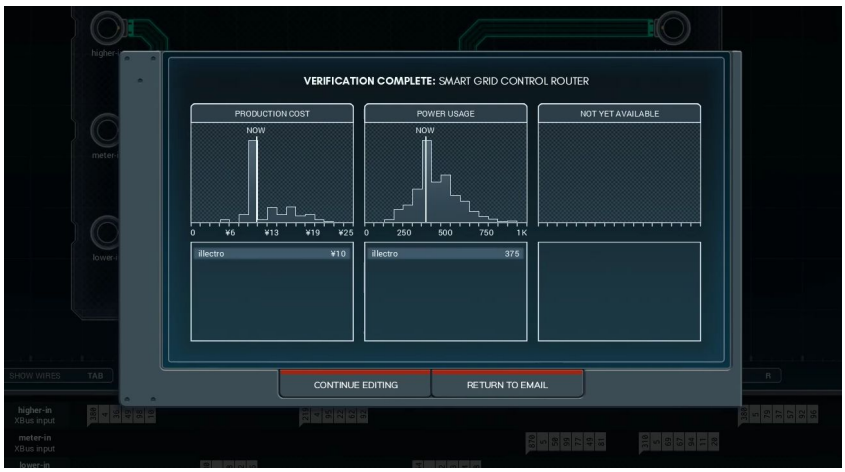
Niveau Personnal Sandwich Assembler

La difficulté ici est d'abord la compréhension du langage de développement (Assembleur), ainsi que du système d'entrée/sortie sous forme de signaux électriques. Le jeu comporte une documentation technique d'une trentaine de pages, ce qui place le jeu dans une catégorie à part de HardCore gamers (Impossible de jouer sans avoir lu et compris la documentation, et certaines pages sont en mandarins…)



Quelques pages de documentation de Shenzhen I/O

En plus de la difficulté initiale (langage, signaux, documentation), le jeu propose à chaque fin de niveau, des statistiques sur les performances du joueur, et les inclus dans un graphe représentant les statistiques globales des autres joueurs. Ce qui apporte une dose supplémentaire de challenge en plaçant le joueur indirectement en compétition avec les autres joueurs. Mais la difficulté apportée est parfois si importante, qu'après plusieurs heures de réflexion, et après avoir enfin réussi un niveau, lorsqu'arrive cet écran de statistiques et que l'on comprend que notre performance est plutôt médiocre, on se dit qu'on pourrait faire mieux, mais à quoi bon s'acharner ?



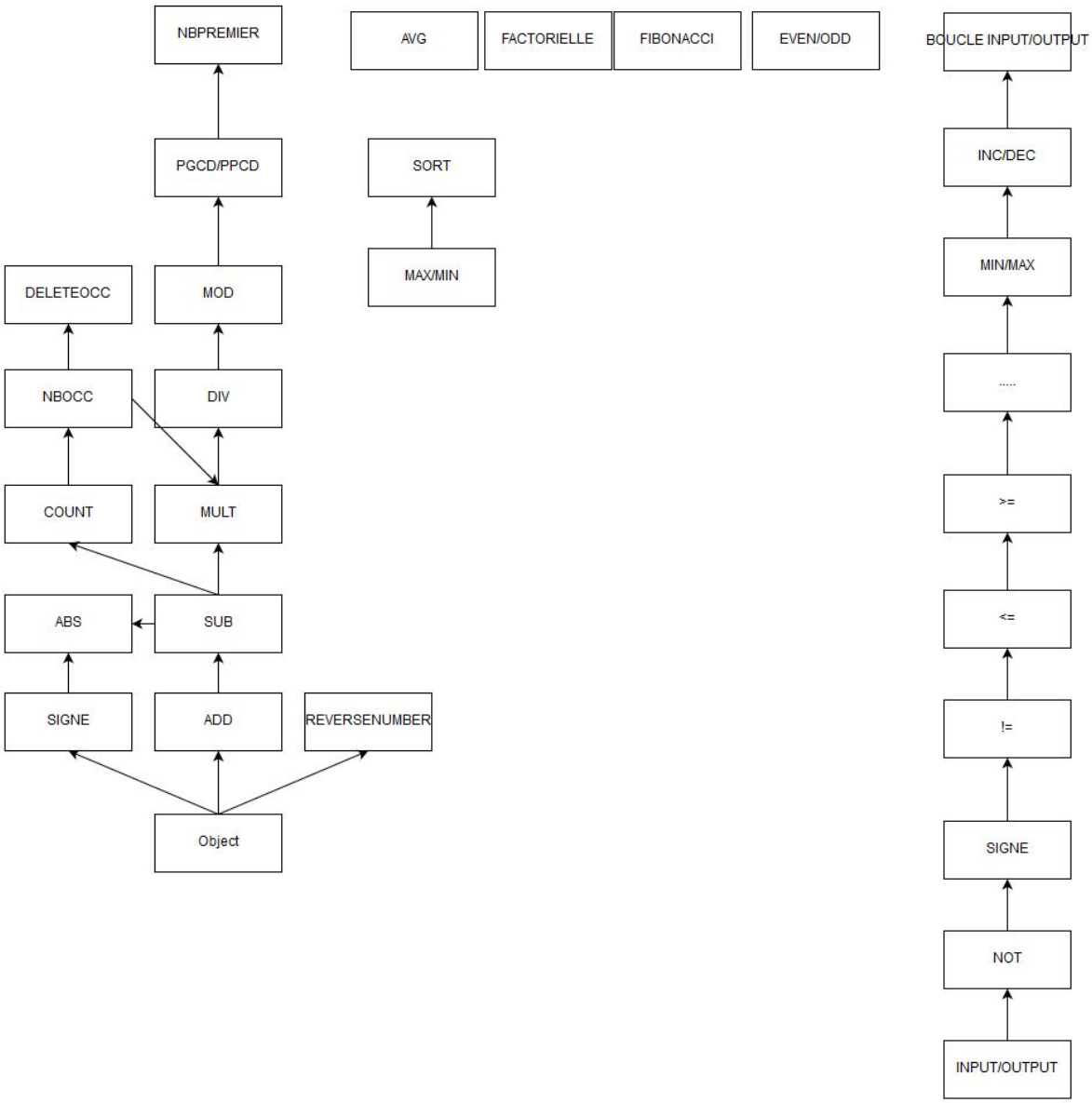
Statistiques apparaissant à la fin d'un niveau

Synthèse

Nom	**Public visé**	**Difficulté**	**Nombre de blocs / éléments de code à disposition**	**Manière de coder**	**Type de puzzles**
Human Resource Machine	Joueurs	Moyenne	~15 commandes	Par blocs	Gestion d'entrées/sorties (nombres ou lettres), programmation par blocs.
CodinGame	Développeurs / Entreprises	N/A	Illimité	Par lignes de code	Libre, au choix du créateur.
Shenzhen I/O	Joueurs aguerris	Élevée	~10 processeurs ~25 commandes	Par lignes de code	Gestion d'entrées/sorties (impulsions électriques) programmation de puces électroniques.
Hour of Code	Élèves (utilisation en classe) ou novices sans limite d'âge	Faible	Illimité	Par blocs	Libre, au choix du créateur. Pour chaque niveau, le créateur choisit les blocs et la mécanique de gameplay (Déplacement d'un personnage ou autre)
Code en blocs (notre jeu)	Elèves de collège (niveau 3ème) *Lycée, niveau première*	Faible à Moyenne	Une vingtaine *8 ont été implémentés.*	Par blocs	Gestion d'entrées/sorties (nombres)

Annexes

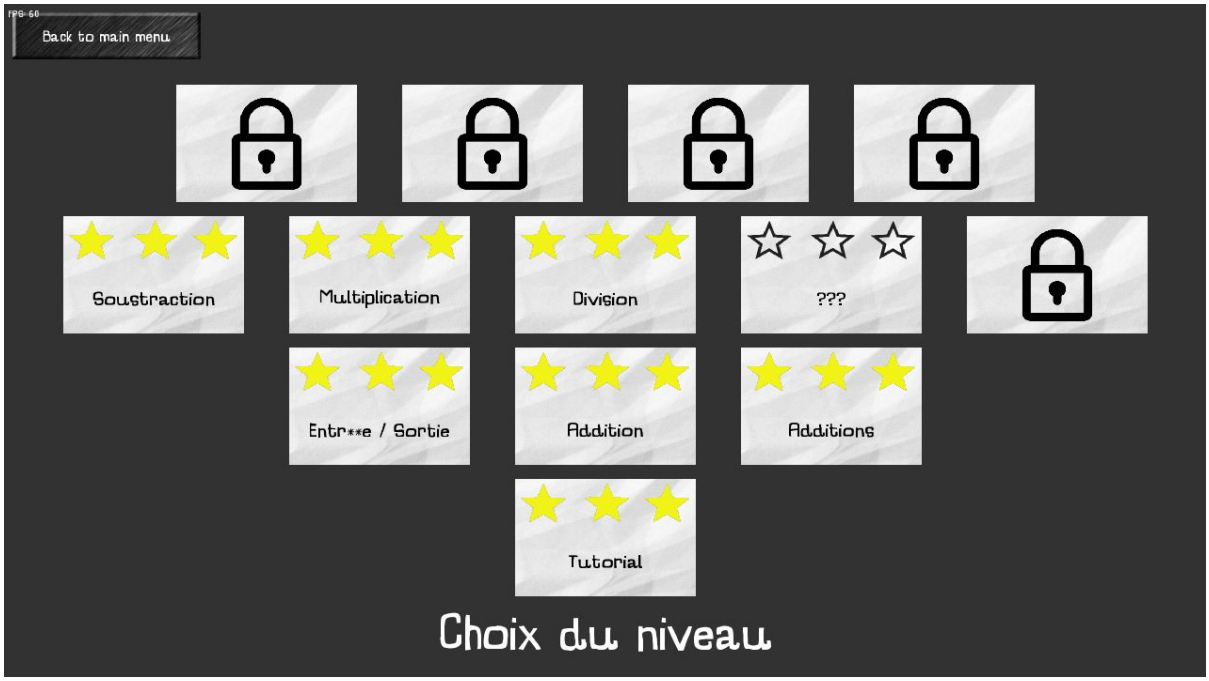
Liste des niveaux (préliminaire)



Arbre des niveaux du jeu

Le diagramme ci-dessus représente une liste non exhaustive des niveaux qui seront présents dans le jeu. La représentation sous forme d'arbre permet d'organiser les niveaux selon une suite logique selon la difficulté mais aussi les blocs disponibles et notions apprises. Ledit arbre sera complété ultérieurement.

Liste des niveaux (actuel)



Liste des blocs (préliminaire)

PRIMITIVES A GAGNER

ADD	SUB	MULT	DIV	MOD	INC
DEC	COUNT	AVG	EVEN	ODD	SORT
MIN	MAX	ISZERO	ISNUMBER	ISLETTER	ABS
NOT*LABEL*	SIGNE	<=	>=	!=	

PRIMITIVES

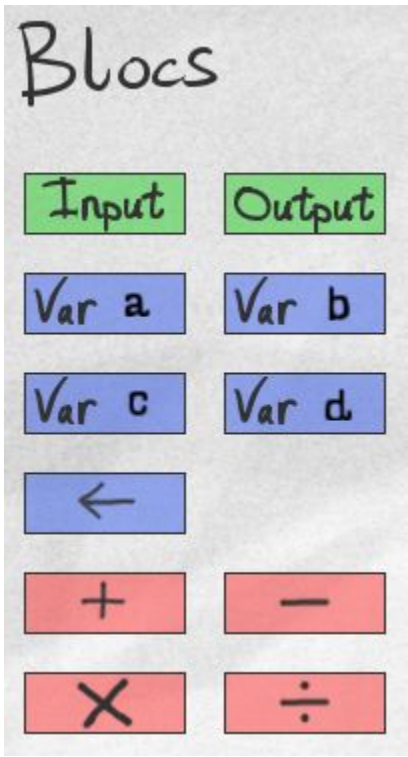
IF/ELSE/FI	FOR	WHILE	DO/WHILE	BREAK	SWITCH/CASE
OUTPUT	INPUT	FUNCTION:NAME	VAR *LABEL*	JUMP COND	JUMP *LABEL*
COPY *NAME* *VAL*	CURRENT	EOS			

<	>	=	AND	OR	

Liste des blocs disponibles pour le joueur

Il s'agit là encore d'une liste sujette à modifications de l'ensemble des blocs présents dans le jeu. La première partie concerne les blocs que le joueur gagnera au fil du jeu en les codant lui-même et la seconde partie liste montre les blocs primitifs dont il aura besoin pour réaliser les niveaux.

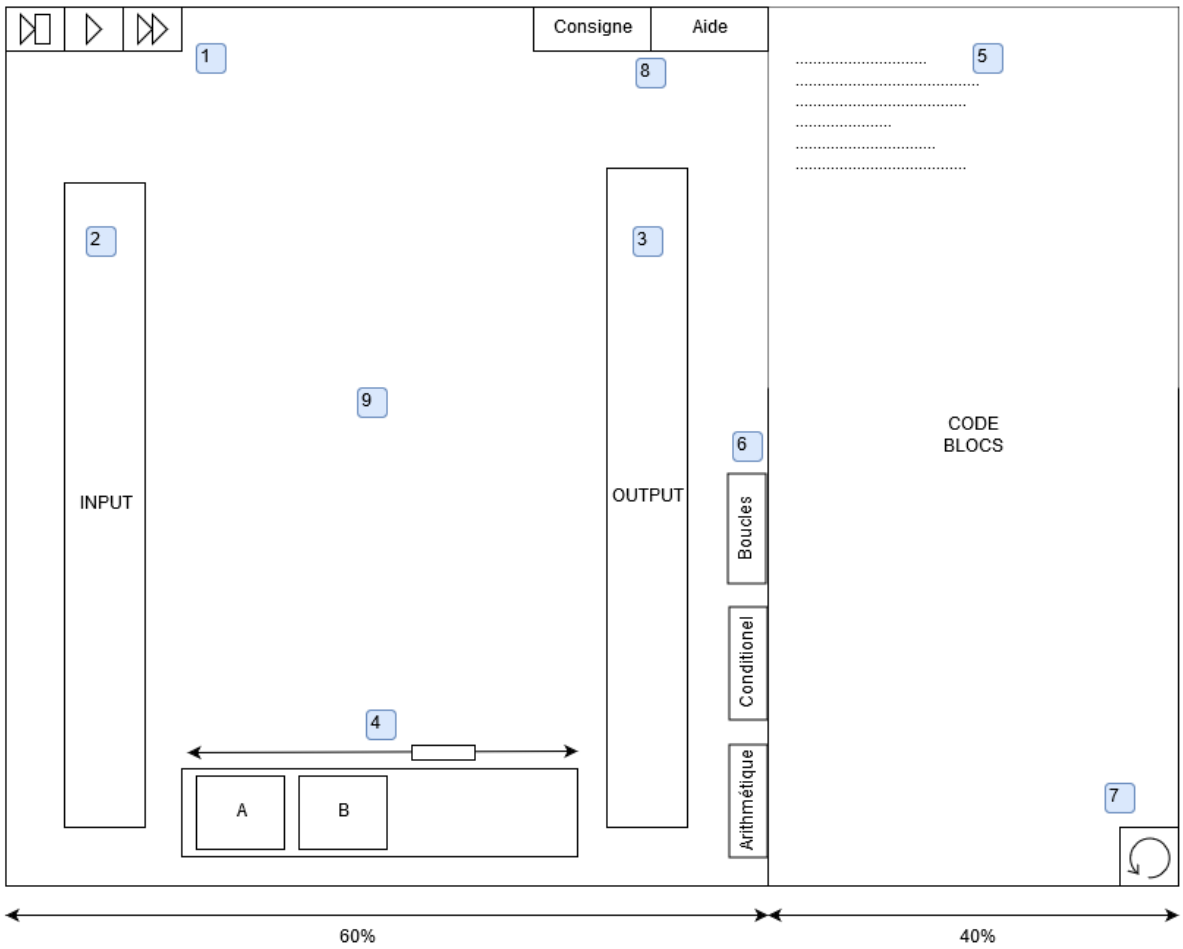
Liste des blocs (actuel)



Par manque de temps, les blocs disponibles ont été revus à la baisse. Et les blocs algorithmiques n'ayant pu être implémentés, le joueur ne peut pas obtenir de nouveaux blocs en les créants. Nous avions prévu de donner au joueur le bloc multiplication après avoir réalisé une multiplication à la main avec une série d'additions dans une boucle ; la boucle n'ayant pu être implémenté, le bloc multiplication est donné au joueur dès le début du jeu.

Interface utilisateur

Mock-up design UI



Explication de l'IHM:

1 - En haut de la fenêtre nous avons 3 boutons:

Le premier (à gauche) sert à visualiser l'exécution de l'algorithme pas à pas, chaque clic sur le bouton exécutant l'instruction en cours.

Le deuxième (au centre) sert à exécuter l'algorithme dans le but de tester le code du joueur et de le voir se dérouler entièrement.

Le troisième (à droite) sert à exécuter l'algorithme avec un rythme rapide, sans visualiser toutes les étapes du code, afin d'arriver au résultat en quelques secondes.

2 - Dans cette "liste défilante vers le haut", nous voyons les entrées. Il peut s'agir de nombres ou de lettres, qui serviront de paramètres pour l'algorithme à créer.

3 - Dans cette "liste défilante vers le bas" nous avons les sorties, ils correspondent aux sorties que doit générer l'algorithmes (comme des nombres, Vrai/Faux, etc…)

4 - En bas sont contenues toutes les variables utilisées lors de l'exécution du programme. Si leur nombre est conséquent, on peut les faire défiler horizontalement.

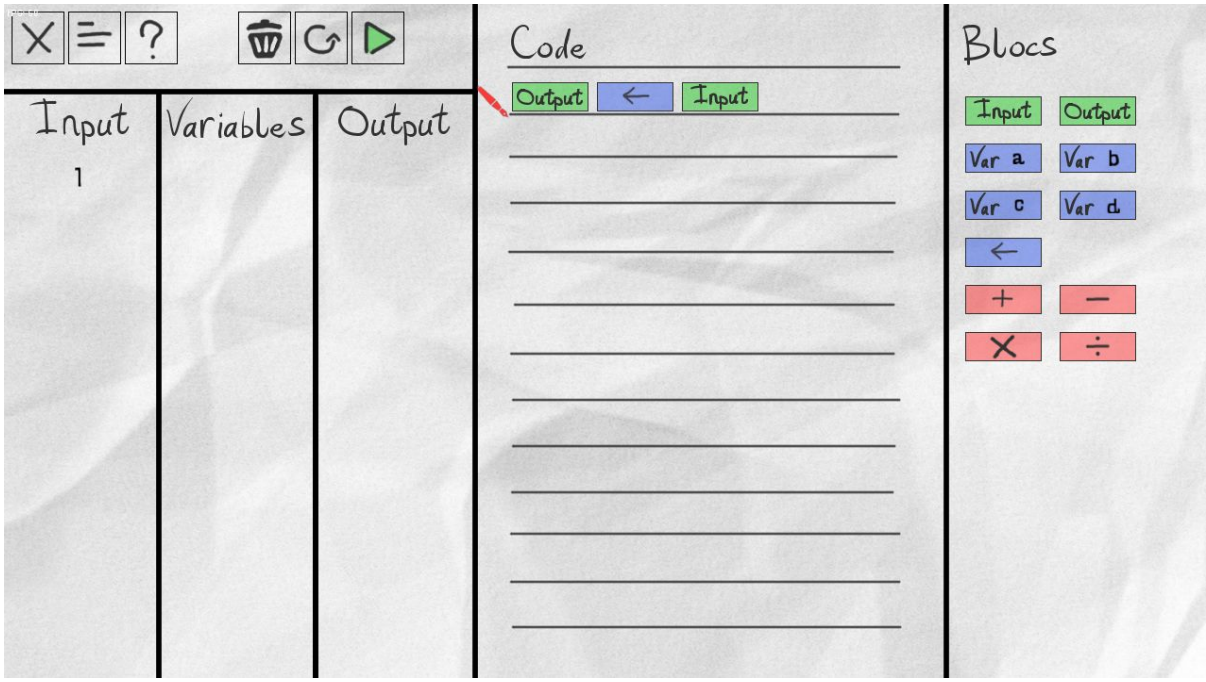
5 - Cet espace contient le code du joueur, composé de blocs de programmation.

6 - Onglets servant à accéder aux blocs disponibles pour la programmation, ouvrant un panneau latéral, contenant les blocs du type de l'onglet (Arithmétique : ADD, SUB, MULT… ; Conditionnel : IF, ELSE, AND, OR ... ; Boucles : WHILE

7 - Ce bouton fait en sorte que le programme du joueur boucle automatiquement après exécution.

8 - Ces deux boutons permettent de rappeler les instructions/consignes de l'algorithme à créer ainsi qu'un exemple/une aide pour le joueur.

9 - Le centre de la fenêtre affiche toutes les informations utils pendant que le joueur joue de plus il affiche une animation montrant l'exécution du code du joueur.



Interface graphique finalement intégrée au jeu. Tous les boutons du menu ont été regroupés en haut à gauche, les Input, Output et Variables ont été regroupés en Workspace, suivis de la CodePage, bien centrale, et de la liste des blocs disponibles dans le niveau actuel.