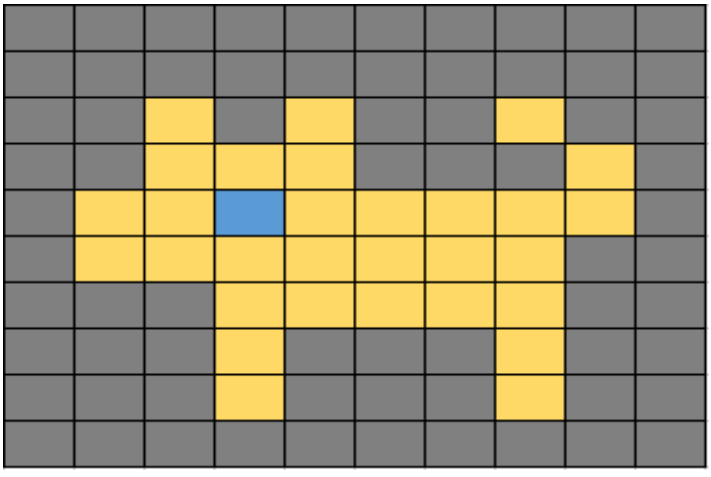
Université **T**oulouse **III** – **P**aul **S**abatier

Département Mathématiques

Formation **S**tatistique et **I**nformatique **D**écisionnelle

RAPPORT DE PROJET

Création d'un site web à partir d'une base de données permettant de jouer au jeu du Pixel Factory.



Mise à jour le 02.05.2020

Théo SACCAREAU et **Stévan STRICOT**

Sous l'encadrement de :

José G. MORENO et **Karen PINEL-SAUVAGNAT**

Résumé

Ce document est un rapport de projet réalisé au cours de l'année universitaire 2019-2020 par Théo SACCAREAU et Stévan STRICOT, étudiants en 3e année de licence en Statistique et Informatique Décisionnelle à l'université Toulouse III – Paul Sabatier.

Le but du projet est d'apprendre la programmation web en manipulant des données stockées dans une base de données. Le site web créé doit permettre de jouer au jeu du Pixel Factory. Ce jeu consiste à reproduire à l'identique une figure composée de petits carrés colorés par différentes couleurs.

Ce rapport permet de retracer l'évolution chronologique du projet en détaillant les choix effectués, les méthodes employées et le résultat obtenu. On expliquera également les difficultés rencontrées et la manière dont nous les avons surmontées.

Mots-clés : Base de données, site web, Pixel-Factory, partie, joueur, niveau, figure.

Table des matières

Table des figures

Liste des tableaux

Introduction

Dans le cadre de notre formation en 3e année de licence en Statistique et Informatique Décisionnelle (SID) à l'Université de Toulouse III – Paul Sabatier, nous avons dû réaliser un site web reposant sur une base de données. Ce projet fait partie du module ELMAB6F1 : Projet site Web avec BDD.

Le site web à créer doit permettre de pouvoir jouer à un jeu. Le jeu à réaliser a été tiré au sort lors de la première séance. Notre groupe s'est vu attribuer le jeu du « Pixel Factory ». Le but de ce jeu est de reproduire à l'identique une figure composée de petits pixels colorés par différentes couleurs.

Ce projet a plusieurs objectifs, il doit nous offrir la possibilité, à nous, étudiants, de découvrir plusieurs notions qui nous seront utiles plus tard, à savoir :

- Revoir les acquis du semestre 5 en base de données et programmation
- Approfondir les connaissances en base de données grâce à l'enseignement de nouvelles notions
- Apprendre de nouveaux langages de programmation (notamment ceux de la programmation web : HTML5, JavaScript, etc)
- Réussir une autoformation : nous avons dû utiliser des technologies non présentées en cours (le langage CSS)
- Savoir comment raisonner face à un problème complexe et comment le résoudre
- Améliorer notre autonomie et nos prises d'initiatives
- Concevoir un exemple concret d'utilisation et de gestion d'une base de données.

Nous avons donc pensé et créé dans sa totalité notre version de ce jeu. Cette réalisation est allée de la réflexion du schéma de la base de données à la mise en forme du site web grâce au langage CSS en passant par la définition de certaines règles du jeu.

Nous avons donc dû procéder méthodiquement afin de mener à bien notre projet. Pour cela, nous avons tout d'abord dû nous répartir les tâches. Cette répartition s'est faite de manière assez naturelle. Stévan Stricot est responsable du développement de la base de données, les tables et les données se trouveront donc sur son compte universitaire Oracle.

C'est donc lui qui est, ce qu'on appellera par la suite, « l'user 1 » ou « l'user base de données ». Bien évidemment, cela ne veut pas dire qu'on n'a pas échangé durant la conception de la base ou que toute la base a été créée par une seule personne, le travail à effectuer a été réparti équitablement. Théo Saccareau était donc lui responsable du développement de la partie web. Il est « l'user 2 » ou encore « l'user Web ».

En plus de s'être réparti les tâches, afin d'avoir un avancement progressif et de réussir à respecter la date limite de rendu, nous avons établi un ordre de développement. Pour établir cette organisation nous n'avions pas une grande marge de manœuvre puisque, par exemple, la base de données ne peut être créée qu'une fois que le schéma « entité-association » a été validé. Nous avons commencé par établir le schéma « entité-association », puis une fois validé, nous l'avons transformé en schéma relationnel permettant alors de créer les tables de la base. Après avoir inséré les données, nous nous sommes concentrés sur la réalisation des procédures, fonctions, triggers et packages du langage PL/SQL. L'un des choix importants que nous avons effectué a été de passer au développement de la partie web uniquement lorsque la base de données avait été totalement finalisée (toutes les données insérées, toutes les procédures et fonctions créées, code commenté, etc.). On souhaitait éviter au maximum de revenir modifier le code (par exemple pour le commenter des semaines après l'avoir écrit). Bien entendu, cela n'a pas empêché qu'on y fasse quelques modifications suite à des bugs repérés lors de son utilisation à partir du web. La dernière étape du projet a donc été le développement de la partie web à savoir la création des pages HTML, la mise en page de celle-ci grâce au CSS, l'interaction web-base de données via SQLAlchemy et le langage Python, rendre des pages dynamiques grâce au JavaScript et la bibliothèque JQuery, etc. Pour ne pas se retrouver débordés lorsque la date de fin arrivait, nous nous sommes également appuyés sur le planning que nous ont donné nos professeurs et qu'ils nous conseillaient de suivre.

Nos priorités à travers ce projet étaient bien évidemment dans un premier temps de respecter les consignes imposées par le sujet mais aussi d'essayer au maximum de faire preuve d'initiative et de rendre notre site encore plus proche de ceux que l'on peut trouver sur des sites de petits jeux en ligne comme le Pixel Factory.

Afin de retracer l'évolution de ce projet, nous vous proposons de voir dans un premier temps les choix que nous avons effectués pour la partie base de données. Une fois que nous vous aurons présenté notre schéma « entité-association », le schéma relationnel puis les caractéristiques de la base (tables, attributs, contraintes, procédures, fonctions, triggers, etc), nous verrons les choix effectués dans la partie web.

1 La base de données

1.1 Schéma entité-association = Modèle Conceptuel de Données

La réalisation du schéma « entité-association » a constitué la première étape de notre projet. Constituant la base de tout le projet, il devait être validé par le corps enseignant pour pouvoir poursuivre la suite du projet. C'est notre deuxième version du schéma qui a été validée, la voici :

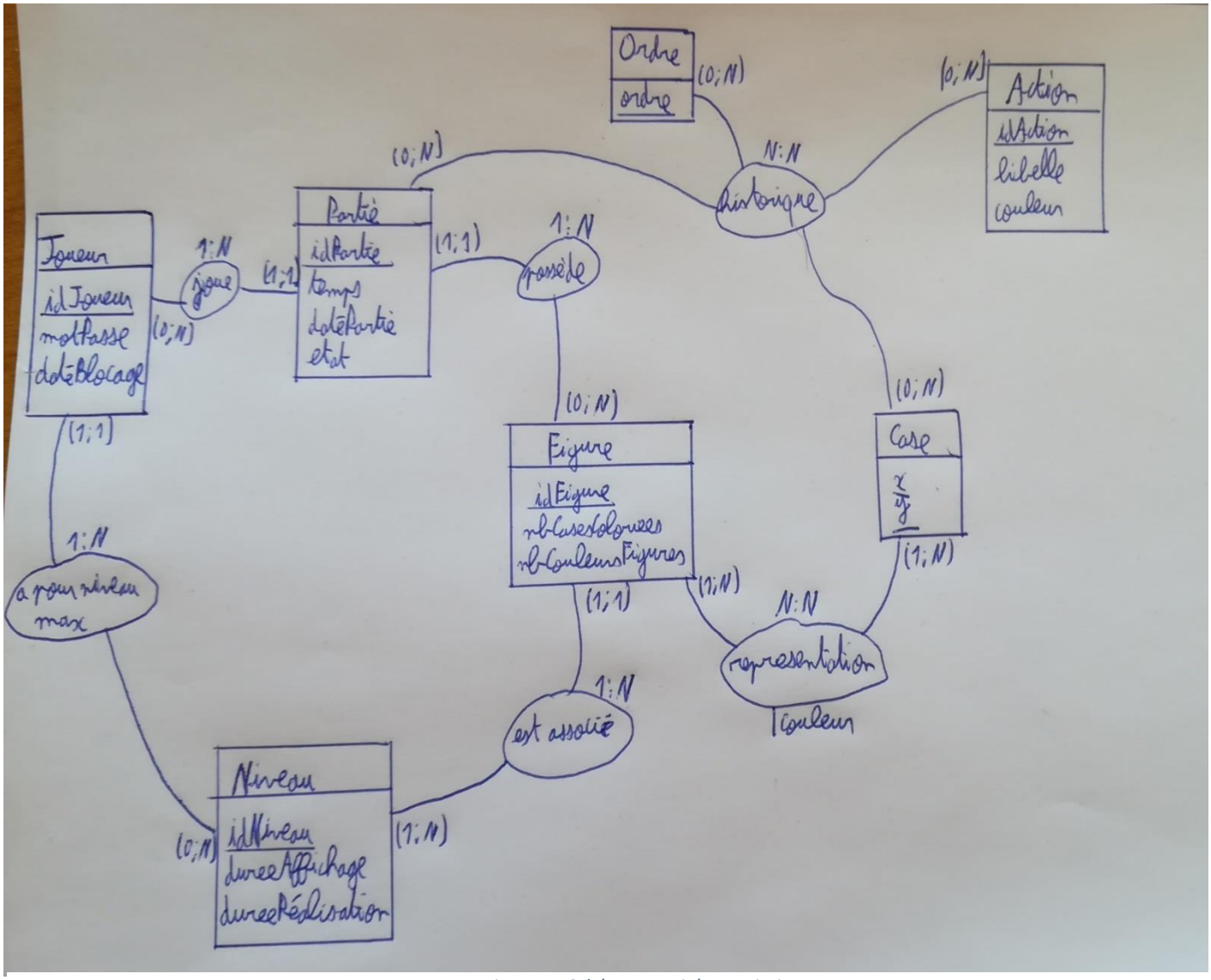


Figure 1 - Schéma « entité-association »

Explication :

- Joueur :

L'entité Joueur a comme attribut l'identifiant 'idJoueur', le mot de passe 'motPasse' et une date de blocage éventuelle 'dateBlocage'. Cette dernière est nécessaire pour bloquer un joueur pendant quatre heures lorsqu'il perd cinq parties en moins d'une heure. L'identifiant correspond au pseudo dans la partie web, il constitue la clé de cette entité.

Elle est reliée avec l'entité Niveau par l'association "a pour niveau maximal" qui nous permet de savoir le niveau maximal des figures auxquelles à accès le joueur. Remarque : lors du passage au modèle logique des données (MLD), il y aura un attribut 'niveauMax' dans l'entité Joueur car c'est une relation de type 1-N.

- Niveau :

L'entité Niveau contient donc l'identifiant du niveau 'idNiveau' (constitue la clé de l'entité), la durée d'affichage de la figure à réaliser (lorsque le joueur demande l'aide) 'dureeAffichage' et la durée de réalisation que possède l'utilisateur pour faire le dessin demandé 'dureeRealisation'.

- Partie :

L'entité Partie possède comme attributs un identifiant 'idPartie', le temps de réalisation 'temps', la date et l'heure à laquelle la partie s'est finie 'datePartie', l'état de la partie 'etat'. L'état de la partie permet de savoir si elle est réussie (« succes ») ou perdue (« echec »). La clé de l'entité est l'identifiant 'idPartie'.

Cette entité est liée bien évidemment à celles de Joueur et de Figure. Cela donne donc lors du passage au MLD l'apparition de 'joueurPartie' et 'figurePartie' dans la relation Partie car ce sont deux associations de type 1-N.

- Figure :

L'entité Figure possède trois attributs : l'identifiant 'idFigure', le nombre de cases que l'utilisateur doit colorier 'nbCasesColoriees' et le nombre de couleurs différentes que la figure comporte 'nbCouleursFigure'. La clé est une fois de plus l'identifiant 'idFigure'.

Cette entité est associée à l'entité Niveau, ce qui fera apparaître dans la relation Figure, lors du passage au MLD, l'attribut 'niveauFigure' (la relation est de type 1-N).

- Case :

L'entité Case contient les attributs x et y qui correspondent aux cordonnées des cases de la grille (numéro de case). Ils constituent la clé.

L'association « Representation » entre les entités Case et Figure est un type N-M. Lors du passage au modèle relationnel, elle deviendra une relation. Elle contiendra comme attributs les clés des entités Figure et Case c'est-à-dire : 'idFigure', 'x' et 'y'. Il faudra également ajouter l'attribut 'couleur' qui est l'attribut de l'association.

Cette relation permettra d'avoir la représentation de la figure à réaliser puisqu'elle stockera pour toutes les cases ('x', 'y') la bonne couleur associée ('couleur'). L'attribut 'idFigure' permettra de savoir à quelle figure la case est associée. La clé de la relation sera constituée des attributs 'idFigure', 'x' et 'y'.

- Action :

L'entité Action a comme attribut un identifiant 'idAction', un libellé 'libelle' pour décrire ce que fait l'action, et une couleur 'couleur'. La clé sera l'identifiant 'idAction'.

- Ordre :

Cette entité a un seul attribut (c'est donc la clé). Cet attribut c'est 'ordre' il permettra d'avoir les actions dans l'ordre pour l'historique.

L'association « Historique » relie les entités Partie, Case, Action et Ordre. Elle est de type N-M, ainsi lors du passage au MLD, elle deviendra une relation. Elle contiendra comme attributs les clés des entités Partie, Case, Action et Ordre c'est-à-dire : 'idPartie', 'x', 'y', 'idAction' et 'ordre'.

Elle stockera les informations permettant de faire un replay d'une partie. En effet, 'ordre' nous permettra de rejouer les actions dans le même ordre que le joueur. De plus les cordonnées 'x' et 'y' nous indiqueront la case où s'est effectuée l'action (qui sera elle déterminée grâce à l'attribut 'idAction'). Pour finir, l'attribut 'idPartie' nous indiquera tout naturellement la partie concernée. La clé sera constituée de tous les attributs.

1.2 Schéma relationnel = Modèle Logique de Données (MLD)

Le Modèle Conceptuel de Données (MCD) correspond au schéma « entité-association ». Il ne peut pas être implanté dans une base de données sans modification.

Il est obligatoire de transformer ce modèle. On dit qu'on effectue un passage du modèle conceptuel de données vers le modèle logique de données.

Le Modèle Logique de Données est également appelé modèle (ou schéma) relationnel (lorsqu'on travaille avec une base de données relationnelle).

Après ce passage, le MLD pourra être implanté dans une base de données relationnelle.

Pour réaliser ce changement, il faut suivre dans l'ordre ces trois règles d'or.

- Règle 1 :

A chaque entité est associé un schéma de relation composé de tous les attributs de l'entité. La clé de l'entité devient la clé de la relation.

- Règle 2 :

Si dans une association A, il existe une entité E pour laquelle la cardinalité du couple (E,A) est (0,1) ou (1,1) (autrement dit l'association est donc de type 1-1 ou 1-N), alors on ajoute dans le schéma de la relation R (celle qui traduit E) tous les attributs de l'association ainsi que la clé de chacune des autres entités de l'association.

- Règle 3 :

Si dans une association A, il n'existe pas une entité E pour laquelle la cardinalité du couple (E,A) est égale à (0,1) ou (1,1) (autrement dit l'association est de type N-M), alors on crée un nouveau schéma de relation contenant une clé de chaque entité participant à l'association ainsi que tous les attributs de l'association.

Remarque : Il n'y a aucune règle pour déterminer la clé de la nouvelle relation.

Une fois qu'on a appliqué ces règles à notre modèle conceptuel de données, on obtient le modèle relationnel suivant :

Joueur (idJoueur, motPasse, #niveauMax, dateBlocage)

Partie (idPartie, temps, datePartie, etat, #joueurPartie, #figurePartie)

Niveau (idNiveau, dureeAffichage, dureeRealisation)

Figure (IdFigure, nbCasesColoriees, nbCouleursFigure, #niveauFigure)

Action (idAction, libelle, couleur)

Historique (#idAction,#idPartie,x,y,ordre)

Representation (#idFigure,x,y,couleur)

Légende :

- les attributs soulignés correspondent à la clé.

- les attributs précédés du symbole '#' correspondent aux clés étrangères.

A noter que comme les relations Case et Ordre ne servent qu'à être référencées dans Historique et Représentation, on peut donc les supprimer (au niveau du MLD seulement).

1.3 Description de contraintes colonne-table

Lors de la création des tables, il est nécessaire d'exprimer des contraintes d'intégrités. Ce sont des clauses permettant de contraindre la modification des tables, faite par l'intermédiaire de requêtes d'utilisateurs (INSERT, UPDATE, DELETE), afin que les données saisies dans la base soient conformes aux données attendues.

Dans cette partie, nous allons détailler table par table le choix des contraintes que nous avons fait. Il convient de noter qu'avec du recul (c'est-à-dire une fois que nous avons réalisé le développement web) on s'est rendu compte que nous avions été un peu trop précautionneux et que certaines sont inutiles. Cependant, on est parti du principe qu'il vaut mieux en prévoir trop que pas assez.

1.3.1 Table Niveau

Pk_idNiveau : indique que 'idNiveau' est la clé primaire de la table.

Ch_idNiveau : indique que 'idNiveau' doit être compris entre 1 et 4.

Ch_dureeAffichage : indique que 'dureeAffichage' doit être supérieur à 0.

Ch_dureeRealisation : indique que 'dureeRealisation' doit être supérieur à 0.

Nn_dureeAffichage : indique que 'dureeAffichage' ne peut pas être Null.

Nn_dureeRealisation : indique que 'dureeRealisation' ne peut pas être Null.

1.3.2 Table Action

Pk_ idAction : indique que 'idAction' est la clé primaire de la table.

Ch_couleurAction : indique que l'attribut 'couleur' doit avoir une des valeurs de la liste suivante : rouge, orange, vert, violet, bleu, rose. Remarque : 'couleur' peut être Null, cela signifie que la case n'est pas coloriée.

Ch_idAction : indique que 'idAction' doit être compris entre 1 et 9.

Nn_libelle : indique que 'libelle' ne peut être Null.

1.3.3 Table Joueur

Pk_idJoueur : indique que 'idJoueur' est la clé primaire.

Fk_niveauJoueur : indique que 'niveauMax' est une clé étrangère référençant l'attribut 'idNiveau' de la table Niveau.

Nn_motPasse : indique que 'motPasse' ne peut être Null.

Nn_niveauMax : indique que 'niveauMax' ne peut être Null.

1.3.4 Table Figure

Pk_idFigure : indique que 'idFigure' est la clé primaire.

Fk_niveauFigure : indique que 'niveauFigure' est une clé étrangère référençant l'attribut 'idNiveau' de la table Niveau.

Ch_nbCasesColoriees : indique que 'nbCasesColoriees' doit être compris entre 1 et 100

Ch_nbCouleursFigure : indique que 'nbCouleursFigure' doit être compris entre 1 et 6.

Nn_nbCasesColoriees : indique que 'nbCasesColoriees' ne peut pas être Null.

Nn_nbCouleursFigure : indique que 'nbCouleursFigure' ne peut pas être Null.

Nn_niveauFigure : indique que 'niveauFigure' ne peut pas être Null.

1.3.5 Table Partie

Pk_idPartie : indique que 'idPartie' est la clé primaire.

Fk_figurePartie : indique que 'figurePartie' est une clé étrangère référençant l'attribut 'idFigure' de la table Figure.

Fk_joueurPartie : indique que 'joueurPartie' est une clé étrangère référençant l'attribut 'idJoueur' de la table Joueur.

Ch_etat : indique que 'etat' doit valoir « succes » ou « echec ».

Ch_temps : indique que 'temps' doit être supérieur ou égal à 0.

Nn_joueurPartie : indique que 'joueurPartie' ne peut pas être Null.

Nn_figurePartie : indique que 'figurePartie' ne peut pas être Null.

1.3.6 Table Historique

Pk_Historique : indique que (idAction, idPartie, x, y, ordre) est la clé primaire.

Fk_idActionHistorique : indique que 'idAction' est une clé étrangère référençant l'attribut 'idAction' de la table Action.

Fk_idPartieHistorique : indique que 'idPartie' est une clé étrangère référençant l'attribut 'idPartie' de la table Partie.

Ch_xHistorique : indique que 'x' doit être compris entre 1 et 10 ou égal à -1 (c'est lorsque le joueur clique sur un bouton et non sur une case de la grille).

Ch_yHistorique : indique que 'y' doit être compris entre 1 et 10.

Nn_historiqueX : indique que 'x' ne peut pas être Null.

Nn_historiqueY : indique que 'y' ne peut pas être Null.

Nn_ordre : indique que 'ordre' ne peut pas être Null.

1.3.7 Table Representation

Pk_Representation : indique que (idFigure, x, y) est la clé primaire.

Fk_idFigureRepresentation : indique que idFigure est une clé étrangère référençant l'attribut idFigure de la table Figure.

Ch_couleurRepresentation : indique que couleur doit avoir une des valeurs de la liste suivante : rouge, orange, vert, violet, bleu, rose (couleur peut être Null).

Ch_xRepresentation : indique que x doit être compris entre 1 et 10.

Ch_yRepresentation : indique que y doit être compris entre 1 et 10.

Nn_representationY : indique que y ne peut pas être Null.

Nn_representationX : indique que x ne peut pas être Null.

1.4 Liste de triggers et explication de leurs objectifs

Les triggers sont des objets de la base de données. Attachés à une table, ils vont déclencher l'exécution d'une instruction, ou d'un bloc d'instructions, lorsqu'une ou plusieurs lignes sont insérées, supprimées ou modifiées dans la table à laquelle ils sont attachés. En effet, à la différence des procédures, il n'est pas possible d'appeler un trigger : un trigger doit être déclenché par un événement (INSERT, DELETE, UPDATE).

De plus, une fois le trigger déclenché, ses instructions peuvent être exécutées soit juste avant l'exécution de l'événement déclencheur, soit juste après (option BEFORE ou AFTER).

Enfin, un trigger exécute un traitement pour chaque ligne insérée, modifiée ou supprimée par l'événement déclencheur (option FOR EACH ROW) ou alors il exécute le bloc une seule fois (option STATEMENT).

Les triggers ont plusieurs objectifs, ils permettent soit de mettre en place des contraintes complexes, soit de mettre à jour des colonnes dérivées ou alors de générer des événements.

Le sujet nous imposait de réaliser trois triggers. Deux nous étaient imposés. On devait en revanche définir le troisième. Nous allons expliquer en détail leurs objectifs.

Les deux triggers imposés portaient sur la même table : Partie. Nous les avons donc regroupés en un seul : le trigger 't_b_i_Partie'.

Il permet de gérer des éventuelles erreurs avant l'insertion d'une partie. En effet, il permet de vérifier si le niveau choisi par le joueur est un niveau auquel il a accès. Dans le cas contraire, on lève une « erreur applicative » qui empêche la partie de s'insérer dans la base de données et qui empêche donc le joueur de jouer une partie de ce niveau. De plus, ce trigger permet également de vérifier si le joueur n'est pas bloqué. Comme nous l'avons vu précédemment, si un joueur perd cinq parties en moins d'une heure, il se retrouve bloqué pendant quatre heures. Comme pour le niveau, si le joueur commence une partie alors qu'il est bloqué, le trigger lève une exception applicative empêchant la partie de se jouer.

Le troisième trigger, nommé 't_a_u_Joueur' porte sur la table Joueur après un UPDATE. Il permet de s'assurer que lorsqu'on augmente le niveau d'un joueur, le nouveau niveau est égal à l'ancien plus un. Le trigger permet également de s'assurer que le joueur a réussi toutes les figures de l'ancien niveau. Si l'une de ces deux conditions n'est pas respectée, le trigger lève une exception applicative empêchant la modification du niveau.

1.5 Liste de procédures et explication de leurs objectifs

Une procédure permet d'automatiser des actions pouvant être complexes.

Une procédure est en réalité une série d'instructions SQL désignée par un nom. Une fois la procédure créée, il est possible d'appeler celle-ci grâce à son nom. Les instructions de la procédure sont alors exécutées.

Nous avons réalisé plusieurs procédures que nous appellerons depuis la partie web, voici la liste des procédures que nous avons codées :

- inserer_figure
- inserer_joueur
- update_joueur
- inserer_partie
- fin_partie
- inserer_historique
- etat_null

Ces procédures servent à modifier ou insérer des données dans la base. Elles permettent surtout de gérer les EXCEPTIONS, c'est-à-dire les erreurs. Ces erreurs interviennent lorsque l'une des contraintes définies plus haut est violée ou que l'un des triggers déclenche une « exception applicative ».

- 'inserer_figure' :

Commençons par la procédure 'inserer_figure' qui est utilisée pour insérer une figure dans la base. C'est la seule procédure qui ne sera pas utilisée dans le code web puisque les données pour une figure sont insérées préalablement. On ajoute tous les attributs de la table Figure, à savoir son nombre de couleur 'nbCouleurs' et son nombre de cases coloriées 'nbCasesColoriees'. L'identifiant est lui généré de façon automatique à l'aide d'une séquence. Comme la majorité des procédures que nous allons vous détailler par la suite, son but principal est de gérer les exceptions. On vérifie ici si le nombre de cases coloriées 'nbCasesColoriees' est entre 1 et 100 et qu'il n'est pas Null, si le nombre de couleurs composant la figure 'nbCouleurs' est entre 1 et 6 et qu'il n'est pas Null, si la clé étrangère 'niveauFigure' existe (autrement si le niveau qu'on ajoute est bien présent dans la table Niveau) et si l'identifiant 'idFigure' n'a pas déjà été ajouté (DUP_VAL_ON_INDEX). Cette dernière ne devrait jamais arriver vu qu'on utilise une séquence qui s'auto-incrémente. Si l'une de ces contraintes est violée, on attribue un code de retour différent de zéro qu'on récupèrera afin de faire des actions adéquates.

- 'inserer_joueur' :

La procédure 'inserer_joueur' est utilisée pour insérer un joueur dans la base. Le niveau maximal débloqué pour un nouveau joueur est 1 donc dès qu'on insère un joueur, on fait en sorte que le niveau soit 1. Ensuite, grâce à la gestion des exceptions, on répertorie toutes les erreurs qui peuvent intervenir lors de l'insertion d'un nouveau joueur. On vérifie que l'identifiant du joueur n'est pas Null et qu'il ne dépasse pas 19 caractères, même chose pour le mot de passe. On s'assure également que le pseudo (l'identifiant) n'est pas déjà attribué. En fonction des erreurs déclenchées, on retourne un certain code de retour qui nous permettra dans le code Python de gérer la situation. On détaillera cette liste à la fin de cette partie.

- 'update_joueur' :

La procédure 'update_joueur' est utilisée pour modifier les données d'un joueur. En théorie, il est possible de modifier le mot de passe avec cette procédure, mais dans le site web nous n'avons pas donné cette possibilité aux joueurs. Cela fait partie des améliorations possibles que nous avons détectées, nous les verrons en détail plus tard. On a vu auparavant que le troisième trigger portait sur l'UPDATE de la table Joueur. On retrouve donc, dans la

gestion des exceptions, les erreurs qu'il permet d'éviter, pour rappel : vérifier que le nouveau niveau est égal à l'ancien plus un et que le joueur a réussi toutes les parties de l'ancien niveau. Cette procédure permet aussi de contrôler que le nouveau niveau est compris entre 1 et 4 et que le nouveau mot de passe ne dépasse pas 19 caractères et qu'il n'est pas Null. Une fois de plus, on attribue des codes de retour à ces erreurs et on les gère dans le code python.

- 'inserer_partie' :

La procédure 'inserer_partie' est utilisée pour insérer des données dans la table partie. Cette procédure est appelée avant que la partie ne se lance. Effectivement, on a vu que les deux premiers triggers portent sur l'INSERT dans Partie et qu'il permet de bloquer le lancement de la partie si le joueur n'a pas débloqué le niveau demandé ou s'il est bloqué. Si l'un de ces cas arrive, on passe dans la partie EXCEPTION et l'insertion ne se fait pas. Cette procédure veille également à éviter que la clé primaire soit dupliquée, autrement dit que le numéro de partie existe déjà (cette erreur doit bien évidemment ne jamais arriver vu qu'on utilise une séquence, mais comme évoqué précédemment il vaut mieux être trop prévoyant) et que les clés étrangères existent (autrement dit que l'identifiant du joueur et de la figure sont dans la base) ou qu'elles ne soient pas nulles.

Par ailleurs, l'appel de cette procédure se faisant avant que la partie soit jouée, on ne dispose pas des informations sur l'état de la partie, sur le temps de réalisation et la date de fin, on insère donc seulement le joueur concerné et l'identifiant de la figure. L'identifiant est généré automatiquement par une séquence.

- 'fin_partie' :

La procédure 'fin_partie' est utilisée pour compléter les informations de la partie qui n'ont pas pu être renseignées avec 'inserer_partie' car la partie n'était pas encore jouée, c'est-à-dire les attributs 'temps', 'datePartie' et 'etat'. On utilise dans un premier temps l'ordre UPDATE. Ensuite, on calcule le score grâce à une fonction du package 'packageScore' que nous détaillerons plus tard. Si le score que nous retourne la fonction vaut zéro, on modifie l'état de la partie et on le met à 'echec' (on refait un ordre UPDATE). En effet, nous avons fait le choix qu'un joueur qui, au cours de sa partie, abuse de la demande d'aide ou qui valide un trop grand nombre de fois sa figure alors qu'elle est incorrecte verra sa partie considérée comme perdue même s'il a réussi à reproduire la figure à l'identique dans le temps imparti. Pour finir, si cette partie est considérée comme perdue, on doit vérifier que ce n'est pas la cinquième

partie perdue dans l'heure. Si c'est le cas, on appelle la procédure 'update_joueur' vu précédemment et on met à jour l'attribut 'dateBlocage' permettant de bloquer le joueur pendant quatre heures.

- 'inserer_historique' :

La procédure 'inserer_historique' est utilisée afin d'insérer dans la table historique, les actions successives du joueur sur les cases afin de pouvoir faire un replay de la partie. On y ajoute donc les informations suivantes : l'identifiant de l'action 'idAction', l'identifiant de la partie 'idPartie', les coordonnées 'x' et 'y' et le numéro 'ordre' permettant de retrouver l'ordre des coups joués. Ce dernier est ajouté grâce à une séquence. Concernant les exceptions, on contrôle que les clés étrangères 'idPartie' et 'idAction' existent bel et bien, que 'x' est compris entre 1 et 10 (ou -1 pour les boutons 'Aide' et 'Validation'), que 'y' est compris entre 1 et 10, qu'aucun attribut ne soit Null, et que la combinaison 'idAction', 'IdPartie', 'x, 'y' et 'ordre' constituant la clé n'est pas déjà présente (aucun risque vu que l'ordre est généré par une séquence).

- 'etat_null':

Cette procédure est appelée à chaque fois qu'un joueur se connecte. Elle permet de mettre les parties qui ont un état 'NULL' à 'echec'. Si une partie n'a pas son état à 'succes' ou 'echec', cela veut dire que la procédure 'fin_partie' n'a pas été appelée, c'est-à-dire que le joueur a quitté la page ou le site sans attendre la fin de la partie. En plus de modifier l'absence d'état, on actualise également la date (elle a aussi comme valeur Null) en y mettant la date actuelle. Cela peut sembler incohérent vu que la partie a peut-être été jouée il y a plusieurs heures, voire même plusieurs jours. Nous détaillons ce choix dans la partie « 2.4 Choix ». A la fin de cette procédure, étant donné que le joueur a perdu une partie de plus, on doit vérifier qu'il n'a pas perdu plusieurs parties en une heure, si c'est le cas on appelle la procédure 'update_joueur' pour lui modifier l'attribut 'dateBlocage' et le bloquer pendant quatre heures.

Par ailleurs, on laisse le temps à Null. En effet, il nous est impossible d'avoir cette valeur. De plus, cela n'a aucun impact puisque le temps est uniquement utilisé pour calculer le score d'une partie. Or nous verrons dans la partie suivante que le score n'est pas calculé et vaut directement zéro pour toutes les parties ayant leur état à 'echec'.

Pour ne pas être perdu avec tous les codes retour dans la partie web, nous avons décidé d'établir une liste des codes en essayant de les regrouper en plusieurs catégories.

Code	Signification
0	Tout s'est bien passé
1	DUP_VAL_ON_INDEX = Clé primaire déjà existante
11 -> 20	**Clé étrangère (FK)**
11	Identifiant de partie inexistante (Procédure : 'inserer_historique')
12	Identifiant d'action inexistant (Procédure : 'inserer_historique')
13	Identifiant de joueur inexistant (Procédure : 'etat_null')
14	Identifiant de joueur et/ou de figure inexistant(s) (Procédure : 'inserer_partie')
15	Identifiant de niveau inexistant (Procédure 'inserer_figure')
16	Identifiant de partie inexistant (Procédure : 'fin_partie')
21 -> 30	**Erreurs portant sur les CHECK**
21	Attribut 'nbCasesColoriees' n'est pas entre 1 et 100 (Procédure : 'inserer_figure')
22	Attribut 'nbCouleursFigure' n'est pas entre 1 et 6 (Procédure : 'inserer_figure')
23	Attribut 'idJoueur' de taille supérieure à 19 (Procédure : 'inserer_joueur')
24	Attribut 'motPasse' de taille supérieure à 19 (Procédure : 'inserer_joueur' et 'update_joueur')
25	Attribut 'niveauMax' n'est pas compris entre 1 et 4 (Procédure : 'update_joueur')
26	Attribut 'x' n'est pas entre 1 et 10 ou -1 (pour les boutons) (Procédure : 'inserer_historique')
27	Attribut 'y' n'est pas entre 1 et 10 (Procédure : 'inserer_historique')
28	Attribut 'temps' est inférieur à zéro (Procédure :'fin_partie')
29	Attribut 'etat' est différent de 'succes' ou 'echec' (Procédure : 'fin_partie')
31 -> 50	**Erreurs portant sur les NOT NULL**
31	Attribut 'nbCasesColoriees' est NULL (Procédure : 'inserer_figure')
32	Attribut 'nbCouleursFigure' est NULL (Procédure : 'inserer_figure')
33	Attribut 'niveauFigure' est NULL (Procédure : 'inserer_figure') Attribut 'niveauMax' est NULL (Procédure : 'update_joueur')
34	Attribut 'idJoueur' est NULL (Procédure : 'inserer_joueur')
35	Attribut 'motPasse' est NULL (Procédure : 'inserer_joueur' et 'update_joueur')
36	Attribut 'x' est NULL (Procédure : 'inserer_historique')
37	Attribut 'y' est NULL (Procédure : 'inserer_historique')
38	Attribut 'ordre' est NULL (Procédure : 'inserer_historique')
39	Attribut 'idAction' est NULL (Procédure : 'inserer_historique')

40	Attribut 'IdPartie est NULL (Procédure : 'inserer_historique')
41 -> 50	**Triggers**
41	Le nouveau niveau n'est pas égal à l'ancien plus un (Procédure : 'update_joueur')
42	Le joueur n'a pas réussi toutes les figures de l'ancien niveau (Procédure : 'update_joueur')
43	Le joueur n'a pas accès à ce niveau (Procédure : 'inserer_partie')
44	Le joueur est bloqué (Procédure : 'inserer_partie')

Tableau 1 - Tableau indiquant les différents codes de retour des procédures

1.6 Package

Dans le langage PL/SQL, un package permet de regrouper plusieurs procédures ou fonctions ayant des caractéristiques communes.

Ainsi, nous avons fait le choix de créer un package 'packageScore' regroupant une fonction ('Score_Partie') et quatre procédures ('Score_Niveau_1J', 'Score_Journee_1J', 'Score_Niveau_MJ' et 'Score_Journee_MJ') qui permettent toutes d'obtenir différents scores. Ces scores seront utilisés notamment dans les tableaux de Highscores présent sur notre site.

Etudions plus en détail ce que permettent de faire chaque fonction ou procédure.

- 'Score_Partie':

C'est une fonction, elle retourne donc un seul élément de type NUMBER. Elle prend en paramètre l'identifiant d'une partie, calcule son score et le retourne. Pour cela, on différencie dans un premier temps le cas où la partie est gagnée ('etat' = 'succes') ou perdue ('etat' = 'echec'). Dans ce dernier cas, aucun calcul n'est nécessaire, le score est de zéro. En revanche, si la partie est réussie, on récupère le temps de réalisation, le nombre de demandes d'aide et le nombre de validations incorrectes (c'est-à-dire le nombre de validations moins un car la dernière validation est forcément correcte vu que la partie est gagnée). Ensuite, on applique la formule suivante :

*score = nombre d'aide * -100 + (nombre de validation-1) * -50 + temps * -3 + 1000*

A noter, si le score est négatif on le modifie pour le mettre à zéro (la partie sera considérée comme perdue).

- 'Score_Niveau_1J':

C'est l'une des quatre procédures. Elle permet, pour un joueur donné, de récupérer son meilleur score pour chaque niveau. Elle prend donc en entrée l'identifiant du joueur ainsi que quatre variables OUT pour chaque score par niveau. Pour récupérer les meilleurs scores, nous utilisons deux curseurs dont un curseur paramétré.

- 'Score_Journee_1J':

Cette procédure permet, pour un joueur donné, de récupérer son meilleur score pour la journée en cours. Elle prend donc toujours l'identifiant du joueur en paramètre mais qu'une seule variable OUT pour récupérer le meilleur score de la journée. Elle nécessite seulement l'utilisation d'un curseur paramétré.

- 'Score_Niveau_MJ':

La procédure 'Score_Niveau_MJ' permet de récupérer le meilleur score pour chaque niveau pour tous joueurs confondus. On récupère également le pseudo de ces quatre meilleurs joueurs (c'est-à-dire leur identifiant). Elle ne prend donc aucun identifiant de joueur en entrée. Cependant, elle prend quatre variables OUT pour chaque score par niveau et quatre autres pour leur pseudo. Pour récupérer les meilleurs scores et les pseudos nous utilisons deux curseurs dont un curseur paramétré.

- 'Score_Journee_MJ':

Comme vous l'aurez sans doute deviné, cette procédure permet de récupérer le meilleur score de la journée en cours pour tous joueurs confondus ainsi que le pseudo de ce meilleur joueur. Elle ne prend donc aucun identifiant de joueur en entrée. Cependant, elle prend une variable OUT pour le meilleur score de la journée et une autre pour le pseudo. Pour récupérer ces informations, on utilise un curseur paramétré.

Pour finir, on peut noter que toutes ces procédures et fonctions sont « publiques » car elles sont toutes directement appelées par le 'user 2' dans le code web.

1.7 Description des droits par utilisateur de la base de données

Le sujet nous imposait de bien différencier deux « users » : celui responsable de la base de données et celui responsable de la partie web. Nous avons succinctement évoqué cette répartition lors de l'introduction.

L'utilisateur 1 (« user 1 » ou « user BD ») est Stévan Stricot (STS2876A), en tant que créateur de la base de données, il en est l'administrateur. Il possède donc tous les droits.

L'utilisateur 2 (« user 2 » ou « user Web ») est Théo Saccareau (SCT2985A). C'est lui qui, sur le code Python, se connecte à sa base. Le problème c'est que sa base ne contient aucune table et aucune donnée (étant donné qu'elle a été créée sur le compte de l'user 1). Il faut donc que l'user BD donne des droits (ou des privilèges objets) pour permettre à l'user Web d'accéder aux informations dont il a besoin. Nous allons les détailler dans cette partie.

Tout d'abord, il a le droit d'exécuter le package 'packageScore' dans son intégralité. Cela lui permet de récupérer les informations nécessaires pour l'affichage des Highscores grâce aux quatre procédures. Il pourra également utiliser la fonction 'Score_Partie' pour afficher à la fin d'une partie le score réalisé.

Ensuite, il a le droit d'exécuter la procédure 'inserer_joueur', elle sera nécessaire lorsqu'un joueur souhaite se créer un compte.

Il a également le droit d'exécuter la procédure 'update_joueur' pour mettre à jour le niveau d'un joueur lorsqu'il a réussi toutes les parties d'un niveau ou mettre à jour la date de blocage lorsqu'il a perdu cinq fois en une heure.

Il a aussi le droit d'exécuter la procédure 'inserer_partie' vu que c'est dans cette procédure qu'il y a les triggers qui permettent de bloquer un joueur qui souhaite un niveau trop élevé ou qui est bloqué.

Il a le droit d'exécuter la procédure 'fin_partie' afin de compléter les informations de la partie dans la base de données (le temps, l'état et la date).

Il a le droit d'exécuter la procédure 'inserer_historique', elle est appelée à chaque fois que le joueur réalise une action.

Enfin concernant les procédures, il a le droit d'exécuter « etat_null » pour qu'à chaque fois que le joueur se connecte on mette à jour les informations des parties qu'il a quittées de manière non conventionnelle (état et date). Finalement, l'user 2 peut donc exécuter toutes les procédures sauf la procédure « inserer_figure ».

En plus d'avoir des droits sur certaines procédures, l'user 2 a des privilèges objets sur des SELECT portant sur plusieurs tables.

Pour commencer, il a le droit de faire un SELECT sur la table 'Joueur'. Cela est particulièrement utile lorsqu'un joueur se connecte, on doit vérifier si le pseudo existe et si la combinaison pseudo / mot de passe est correcte.

Ensuite, il a le droit de faire un SELECT sur 'Historique'. Cela lui permet de récupérer le nombre de coups d'une partie et l'afficher avant que la partie ne soit rejouée dans une page informative. La deuxième raison est de permettre de récupérer des informations pour le replay. D'ailleurs pour obtenir ces informations, il est nécessaire qu'il ait les droits pour faire un SELECT sur 'Action' car une jointure 'Action'-'Historique' est obligatoire pour récupérer ces données.

Il a aussi le droit de faire un SELECT sur 'Représentation' afin de pouvoir récupérer la liste des couleurs des cases (dans l'ordre) et pouvoir afficher la représentation de la figure (dans l'aide ou au début).

Il a également le droit de faire un SELECT sur 'Niveau' pour récupérer la durée de l'affichage et de réalisation avant le début de la partie et pour l'afficher dans une page informative.

Il a le droit de faire un SELECT sur la table 'Partie' pour récupérer des informations afin de construire le formulaire qui permet de choisir la partie qu'on souhaite rejouer (il est nécessaire d'avoir les droits pour faire un SELECT sur 'Figure' également). Cela permet également de récupérer le temps d'une partie qu'on souhaite rejouer (et on l'affiche dans une page informative avant le replay). Enfin, cela permet d'obtenir la dernière partie jouée.

Pour finir, il a le droit de faire un SELECT sur la table 'Figure' pour plusieurs raisons. Premièrement, pour récupérer le niveau d'une figure. Ensuite, pour récupérer des informations sur une figure (temps de réalisation et temps d'affichage de l'aide) qui seront affichées sur la page informative avant le début d'une partie. Comme vu antérieurement, pour pouvoir créer le formulaire permettant de choisir la partie à revoir il faut faire une jointure 'Partie'-'Figure'. On doit également faire cette jointure pour sélectionner la figure que va réaliser le joueur lors d'une partie (on verra par la suite qu'on ne choisit pas cette figure uniquement aléatoirement, il est donc nécessaire de récupérer un certain nombre d'informations).

1.8 Problèmes rencontrés et solutions apportées

Le premier problème que nous avons rencontré était la réalisation du modèle conceptuel de données et notamment savoir comment créer la table Historique. Bien que notre schéma « entité-association » a été validé au deuxième coup par le corps enseignant, il a fallu plusieurs changements de stratégie pour enfin réussir à trouver un moyen conventionnel permettant de représenter cette table. Pour réussir à aboutir au résultat souhaité par les professeurs, nous avons organisé plusieurs réunions où nous avons échangé nos points de vue sur la manière d'aborder le problème. Les cours du semestre 5 de l'UE « Concepts Fondamentaux des Bases de Données » réalisés par Monsieur MORVAN Franck nous ont été très utiles pour réaliser ce MCD.

Le second problème que nous avons rencontré a été de savoir comment faire pour mettre en place la contrainte imposée par le sujet qui demandait à ce qu'un joueur qui a perdu 5 parties dans l'heure ne puisse plus jouer pendant 4 heures. Au début, nous comptions faire un trigger portant sur l'INSERT de la table Partie. L'idée était de compter le nombre de parties ratées dans l'heure et si ce nombre était supérieur ou égal à 5, alors on empêchait l'insertion de la partie dans la table Partie de la base de données. L'information se trouvant dans la table Partie, le trigger portant lui aussi sur cette table, on avait donc le problème de « table mutante ». Nous nous sommes donc référés aux cours de Madame PINEL-SAUVAGNAT Karen qui nous indiquaient d'utiliser l'astuce du « COMPUND TRIGGER ». Cependant comme le SELECT devait se trouver dans la section "BEFORE EACH ROW", nous nous retrouvions une nouvelle fois bloqués. Face à ce problème, nous avons décidé d'ajouter un attribut dans la base afin de contourner le problème : l'attribut « DateBlocage » dans la table Joueur. Comme nous l'avons vu, cet attribut est éventuellement mis à jour à la fin de chaque partie. Il reste à Null (ou à une ancienne date s'il le joueur a déjà été bloqué) si ce n'est pas la cinquième partie perdue en une heure. En revanche si c'est le cas, on le bloque pendant quatre heures, donc 'DateBlocage' prendre la valeur de SYSDATE (correspond à la date actuelle) plus quatre heures.

Enfin, l'insertion des données pour la table 'Representation' a été un point assez compliqué. En effet, nos grilles contiennent cent cases, ainsi nous devions insérer un tuple pour chaque case soit cent tuples par figure. Sachant que nous avons huit figures, cela impliquait de saisir 800 tupples à la main, ce qui nous paraissait très long. Nous avons donc fait le choix de coder un programme Python pour insérer les tupples via SQLAlchemy.

Lorsqu'on exécute ce programme, il nous donne la liste des figures qui n'ont pas encore de représentation, on sélectionne donc celle qu'on veut choisir. Ensuite, pour chaque case on doit saisir la couleur avec des abréviations (B pour Bleu, Rg pour rouge, Ro pour rose etc). Une fois toutes les cases rentrées on insère dans la BD et on fait un 'COMMIT'. La grande force de ce programme est qu'il est plus rapide que d'insérer les 800 tupples un par un mais aussi on a veillé à envisager tous les cas critiques possibles (erreur de saisie, possibilité d'annulation, choix d'un mauvais identifiant pour une figure, etc) et à ce qu'aucune contrainte ne soit violée :

-> Pas de duplication de clé primaire car on vérifie préalablement que l''IdFigure' n'est pas déjà présent dans la table 'Representation'. De plus, le couple (x,y) est unique (pour la figure 'IdFigure') grâce aux boucles WHILE.

-> Pas de clé étrangère non existante car on vérifie préalablement que l''IdFigure' est bien présent dans la table Figure.

-> Pas d'exception liées aux contraintes des CHECK grâce au dictionnaire.

Pour plus de détail, nous vous encourageons à voir directement le code qui est fortement commenté pour suivre pas à pas notre démarche.

Voici ce qui conclut la première partie portée sur la partie base de données. Concentrons-nous désormais sur la partie Web. Nous verrons dans un premier temps l'architecture globale et l'architecture fonctionnelle avant de regarder plus en détail les structures HTML de nos pages. Nous conclurons cette deuxième partie comme la première en évoquant les problèmes rencontrés et les solutions apportées pour les résoudre.

2 La partie web

2.1 Architecture globale utilisée pour le projet

L'architecture globale correspond à la structure générale inhérente à un système informatique, c'est-à-dire l'organisation des différents éléments du système (logiciels et/ou matériels et/ou humains et/ou informations) et des relations entre les éléments. Voici ci-dessus l'architecture globale pour laquelle nous avons opté pour ce projet :

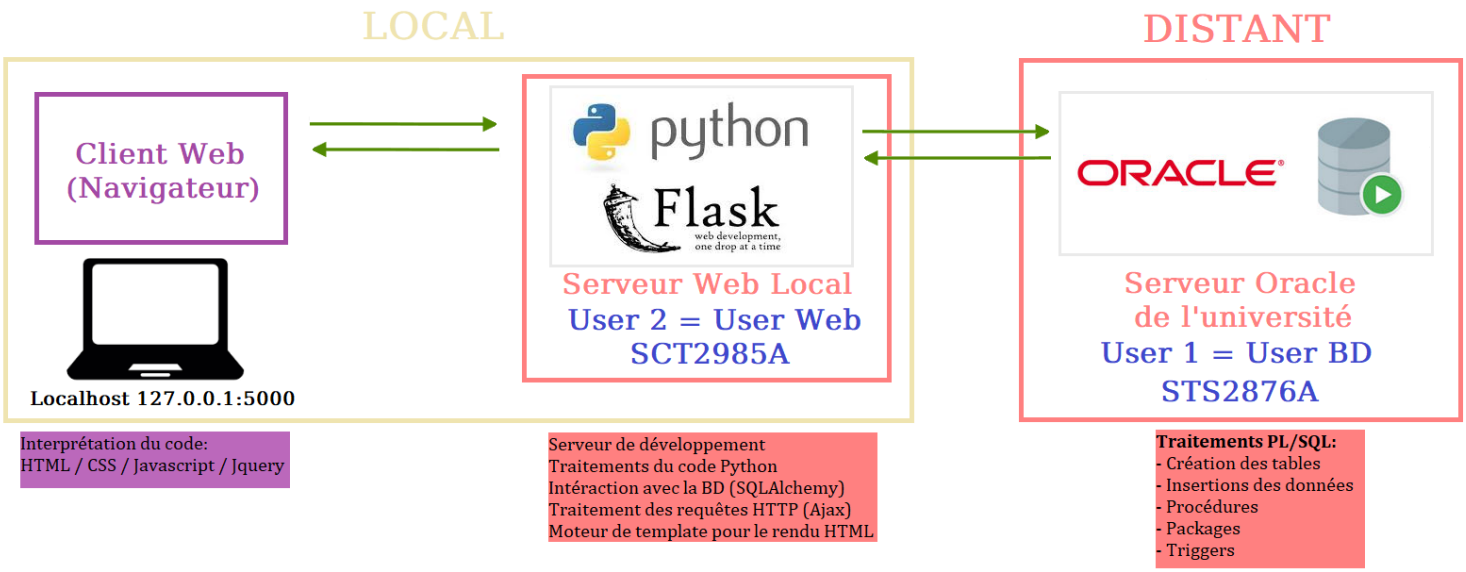


Figure 2 - Schéma de l'architecture globale

Malgré ce qui était prévu au départ, dû à l'arrêt des cours, notre serveur web n'a pas pu être déployé sur le serveur SID. Il est donc uniquement déployé sur un serveur web local.

Toutefois, la base de données est, elle, bel et bien sur un serveur à distance. En effet, nous avons utilisé le serveur Oracle de l'université.

Comme les consignes l'exigeaient, nous avons créé la base de données avec « l'user 1 », c'est-à-dire le compte de Stévan (STS2876A), puis l'accès à la base via le web (avec SQLAlchemy) se fait avec le compte de « l'user 2 », celui de Théo (SCT2985A). Pour que l'user 2 ait accès aux éléments nécessaires au bon fonctionnement nous avons dû faire des droits (privilèges objets), nous les avons déjà détaillés précédemment.

2.2 Architecture fonctionnelle d'une partie monojoueur

L'architecture fonctionnelle correspond à la façon dont les pages de notre site s'enchaînent. On y précise également les technologies utilisées, celles permettant la navigation entre les pages.

Avant de voir en détail le contenu des pages et les fonctionnalités qu'elles offrent aux joueurs, voyons comment elles sont organisées et comment on peut naviguer sur le site.

2.2.1 Transition des pages

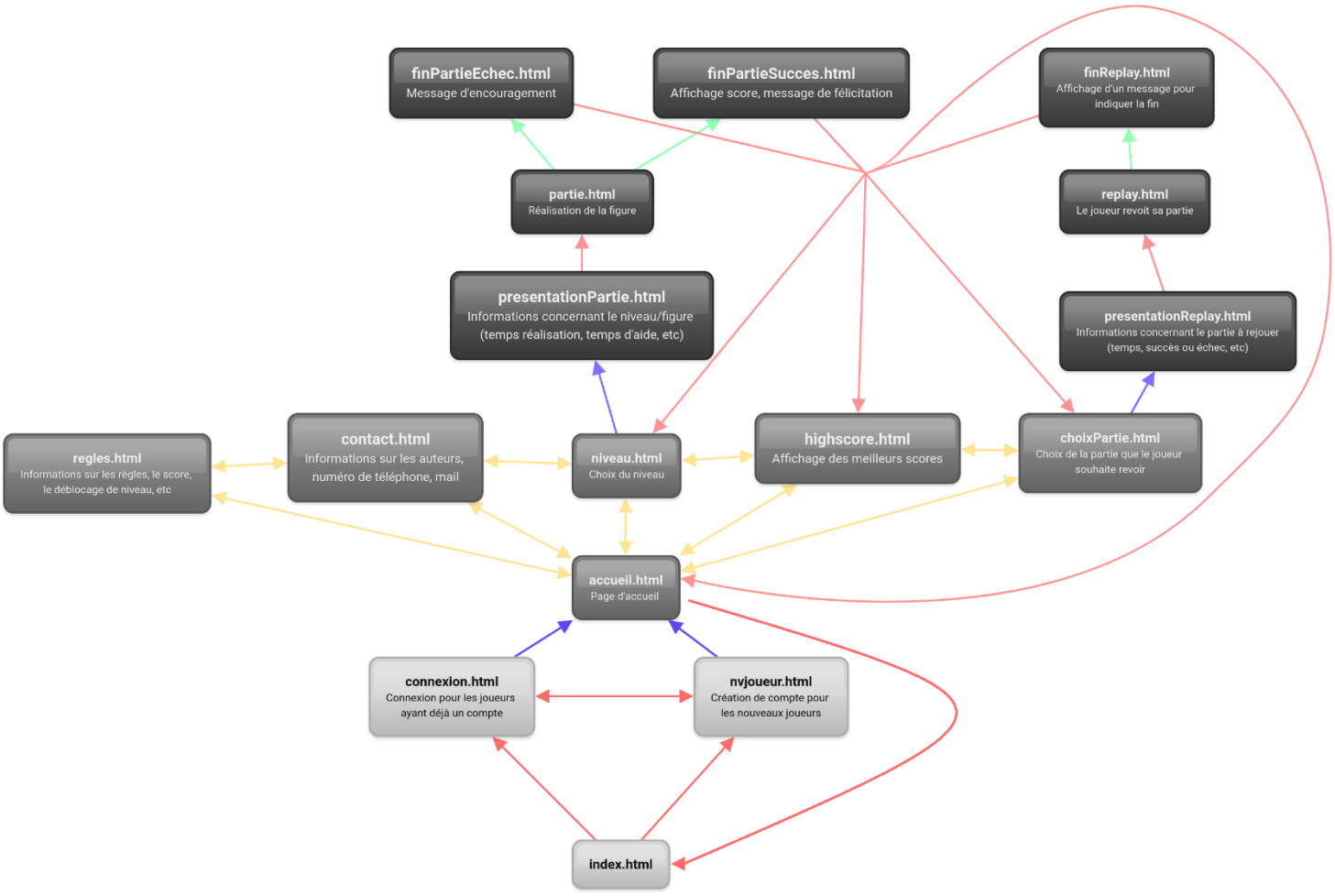


Figure 3 - Schéma de l'organisation des pages du site

Comme on peut le voir sur le schéma ci-dessus, notre site web est constitué de seize pages HTML permettant entre autres de jouer au jeu du pixel Factory ou revoir une partie.

Les flèches colorées indiquent la technologie utilisée pour naviguer entre les pages, il y en a quatre différentes (rouge, bleu, jaune et vert). De plus, on a regroupé les pages en trois catégories (gris clair, gris foncé et noire).

Afin de comprendre la différence entre ces catégories et à quoi correspondent les couleurs, voyons le détail de la légende. Nous en profiterons également pour détailler le rôle et le contenu de chaque page.

2.2.1.1 Couleur des flèches

Les flèches renseignent la méthode utilisée pour changer de page.

- Les flèches **orangées** représentent un changement de page via un bouton HTML.

- Les flèches **bleues** illustrent un passage à une autre page suite à l'envoi d'un formulaire (bouton *<input type="submit">*).

- Les flèches **jaunes** symbolisent une modification de page via un lien du menu. Pour que le schéma reste lisible, on a fait le choix de ne pas les faire tous figurer. Par exemple, on peut passer de la page « règles » à la page « Highscores » grâce au menu alors que ce lien direct est absent dans notre schéma.

- Les flèches **vertes** matérialisent une redirection effectuée grâce au code JavasScript.

2.2.1.2 Couleur des cases

On a regroupé les pages en trois catégories :

- **gris clair** -> trois pages permettant à l'utilisateur de se connecter ou de créer un compte. Catégorie que l'on appellera « Connexion ».

- **gris foncé** -> six pages constituant le corps du site. Elles correspondent aux six actions que peut faire un joueur : aller à l'accueil, voir les règles, consulter les coordonnées des créateurs, joueur une partie, voir les Highscores ou revoir le déroulement d'une partie. Elles comportent toutes un menu permettant de naviguer dans le site. Catégorie « Menu »

- **noir** -> cette catégorie rassemble les pages qui permettent de jouer ou de revoir une partie, c'est la partie « dynamique » de notre site. Ces pages ne possèdent pas de menu car une fois qu'une partie ou un replay est lancé, on ne peut pas faire pause. Catégorie « Dynamique ».

2.2.2 Détails des pages et fonctionnalités

2.2.2.1 Catégorie « Connexion » (gris clair)

- index.html :

Lors de l'arrivée sur le site, via la page « index.html », le joueur à la possibilité (grâce à des boutons HTML) d'aller vers une page permettant de créer un compte ou, s'il en a déjà un, vers une page permettant de se connecter. En cas de mauvais choix, des boutons HTML permettent de naviguer entre ces deux pages.

- nvjoueur.html & connexion.html

Ces pages contiennent un formulaire permettant à l'utilisateur de saisir un pseudo et un mot de passe.

On peut résumer ces deux fonctionnalités (création de compte ou connexion) à l'aide du schéma suivant :

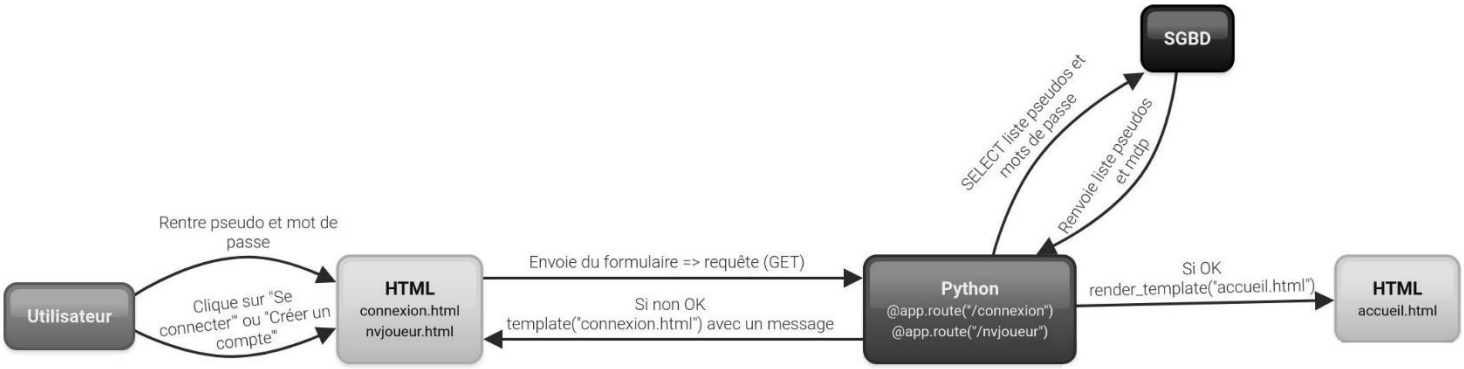


Figure 4 - Schéma illustrant les fonctionnalités "Créer un compte" et "Se connecter"

Grâce au formulaire, l'utilisateur rentre son pseudo et son mot de passe (ou le pseudo et le mot de passe souhaités dans le cas d'une création de compte). Une fonctionnalité HTML, permet d'interdire la saisie de pseudo ou mot de passe de plus de 19 caractères afin de respecter l'une des contraintes fixées dans notre base de données Oracle.

Ensuite, il peut valider sa saisie en cliquant sur le bouton HTML « Se connecter » (ou « Créer un compte »). Ce bouton est de type « Submit », il permet donc d'envoyer une requête de type « GET » contenant les informations renseignées.

Ces informations sont récupérées via le fichier Python (grâce aux commandes « *request.args.get('pseudo')* » et « *request.args.get('pass')* »). Elles sont alors comparées avec la liste des pseudos et des mots de passe que l'on récupère en interrogeant la base de données.

Pour que le code Python renvoie le joueur vers la page d'accueil, il faut :

- dans le cas où il se crée un compte, que le pseudo saisi ne soit pas dans la liste des pseudos déjà attribués.

- dans le cas où il se connecte, que le pseudo soit dans la liste des identifiants et qu'en plus la combinaison pseudo / mot de passe existe. Le fait de faire deux vérifications permet de savoir si c'est le pseudo qui est incorrect ou le mot de passe.

Si ces conditions ne sont pas respectées, le joueur reste sur la page « connexion.html » (ou « nvjoueur.html ») et un message d'erreur apparaît et lui indique ce qui ne va pas.

Remarque : juste avant de passer à la page d'accueil, on ajoute le pseudo dans le dictionnaire « *session* [1]». Dans le cas d'une connexion, on appelle par la même occasion la procédure « etat_null » afin de modifier l'état des parties sans valeur (ni 'succes', ni 'echec') car le joueur a quitté la page directement.

2.2.2.2 Catégorie « Menu » (gris foncé)

- accueil.html :

Une fois l'authentification correcte, le joueur arrive sur la page d'accueil.

Il peut y trouver (comme toutes les pages de cette catégorie) une barre latérale (un « sidebar »). Cette barre latérale contient le menu permettant de naviguer entre les différentes pages, une section contenant des informations sur le projet et enfin un pied de page pour les droits d'auteurs.

Au centre de la page se trouve des explications permettant de découvrir les différentes opportunités qu'offre notre site au joueur.

[1] Le dictionnaire « session » permet de stocker sur le serveur des informations sur le joueur, la partie ou la figure en cours. Ces informations sont donc accessibles à partir de n'importe quelle page du site.

C'est également sur cette page que se trouve le bouton de déconnexion. L'utilisateur est automatiquement renvoyé à la page 'index.html' lorsqu'il clique sur celui-ci.

- regles.html :

Cette page est utile notamment pour les nouveaux joueurs. En effet, on y détaille toutes les règles du site, le joueur peut donc obtenir des informations concernant le but du jeu, le déroulement d'une partie, le calcul du score ou encore le déblocage des niveaux.

- Le but du jeu :

Cette section indique que le joueur doit reproduire à l'identique la figure qui s'affiche. Elle prévient également qu'au bout de cinq parties perdues en une heure, le joueur se trouve bloqué pendant quatre heures.

- Le déroulement de la partie :

On y explique entre autres comment réaliser la figure, le rôle des différents boutons et qu'il y a un temps imparti pour jouer la partie.

- Le score :

On explique au joueur sur quels éléments nous nous basons pour calculer le score à la fin de chaque partie (sans rentrer dans les détails de la formule).

On le prévient également qu'un score négatif (utilisation abusive de l'aide et/ou nombre de validations incorrectes trop élevé) provoque une partie perdue (même si le joueur a réussi à reproduire la figure dans le temps imparti !).

- Les niveaux :

On y détaille comment débloquer un niveau, comment la figure est sélectionnée lorsqu'il a choisi un niveau et l'impossibilité de choisir un niveau auquel il n'a pas accès même s'il apparaît dans la liste.

Remarque : On détaillera tous les choix que l'on a fait dans la partie 2.4.

- contact.html :

Comme dans la majorité des sites web, on a fait le choix d'intégrer une page permettant aux joueurs de nous contacter.

La première partie de la page est un texte d'information sur les créateurs du site.

La deuxième, permet d'obtenir nos numéros de téléphone et nos adresses mails.

- highscore.html :

Cette page comporte deux tableaux de scores.

Le premier tableau, récapitule les meilleurs scores du joueur connecté pour chacun des quatre niveaux.

Le deuxième, permet d'avoir son meilleur score pour la journée en cours (tous niveaux confondus).

En plus d'indiquer les meilleurs scores de ce joueur, nous avons fait le choix de rajouter ces informations pour le meilleur joueur parmi tous les joueurs (score et pseudo). Cette information permet notamment d'avoir un ordre de grandeur et permet au joueur de se comparer aux meilleurs.

Pour comprendre comment sont obtenus ces tableaux de Highscores, nous avons réalisé un petit schéma :

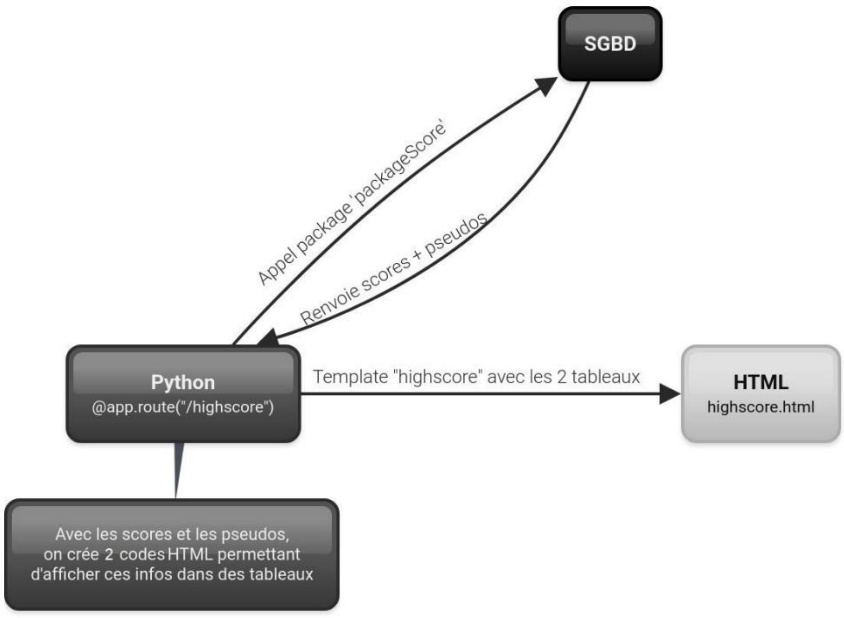


Figure 5 - Schéma illustrant la création des tableaux de "Highscores"

Tout d'abord, le code Python assure une interaction avec la base de données afin de récupérer les différents scores et les pseudos qui seront affichés dans les deux tableaux. Ensuite, une fonction crée, à partir de ces informations, deux codes HTML qui permettent l'affichage des tableaux. Pour finir, on affiche la page « highscore.html » en remplaçant les « content 1 et 2 » par les codes HTML précédemment créés, la commande utilisée est : *render_template("highscore.html",content1=Markup(code_html1),content2=Markup(code_html2)*

Remarques : Les pages 'niveau.html' et 'choixPartie.html' font partie de cette catégorie (dans le sens où elles possèdent un menu) mais nous les détaillerons dans la partie suivante car elles sont fortement liées aux pages de la catégorie « dynamique ».

2.2.2.3 Catégorie « pages dynamiques » (gris foncé) :

**** Jouer une partie*** **:**

Comme on peut le voir sur le schéma fonctionnel, lorsqu'on souhaite jouer une partie, la première page est 'niveau.html'.

- niveau.html :

Cette page permet de choisir le niveau de la partie que souhaite le joueur. Pour nous aider à détailler cette fonctionnalité, nous avons réalisé un schéma :

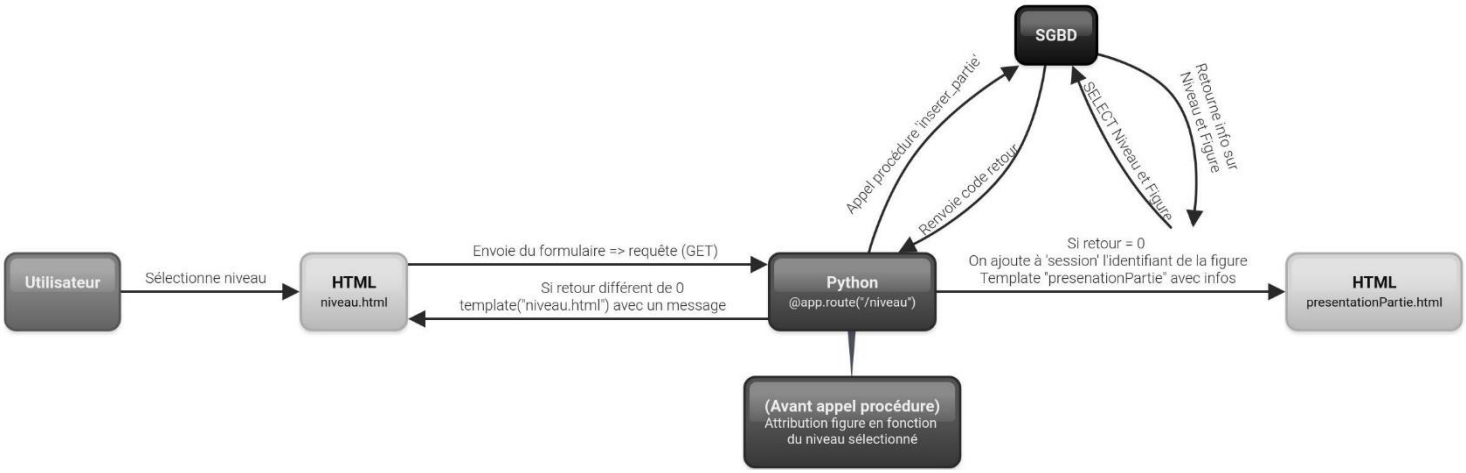


Figure 6 - Schéma illustrant la fonctionnalité "choix du niveau"

Cette page contient un formulaire sous forme d'une liste déroulante. Cette liste comporte les quatre niveaux, le joueur peut donc en pratique sélectionner n'importe quel niveau, même ceux auxquels il n'a pas accès (car non débloqués).

Cependant, lorsqu'il valide son choix de niveau une requête de type « GET » contenant l'identifiant du niveau sélectionné est envoyée. Cet identifiant est récupéré dans le code Python (de la même manière dont on a récupéré le pseudo et le mot de passe). Ce dernier donne la possibilité au code Python d'attribuer une des deux figures du niveau (on verra que le choix de la figure n'est pas tout le temps aléatoire). Ensuite, grâce à l'identifiant de la figure attribuée, on peut appeler la procédure PL/SQL « inserer_partie ». Comme nous l'avons vu dans la première partie, lorsqu'on appelle nos procédures, elles nous retournent des codes permettant de savoir si elles ont bien fait ce qui était attendu.

Par conséquent, si le retour est zéro, tout s'est bien passé, le joueur peut donc jouer sa partie et on le redirige vers la page « presentationPartie.html ». Cette page nécessite des informations concernant le niveau sélectionné et la figure attribuée, c'est pour cela qu'avant de le rediriger, on interroge la base de données pour obtenir ces informations.

En revanche, si le joueur sélectionne un niveau qu'il n'a pas encore débloqué ou qu'il est bloqué parce qu'il a perdu cinq parties en moins d'une heure, alors le code de retour sera différent de zéro, la partie ne se lancera pas et le joueur restera sur la page « niveau.html » avec un message qui s'affichera en indiquant le type d'erreur.

Comme expliqué auparavant, cette page fait partie de la catégorie « Menu » (gris clair), elle possède donc un menu. Si le joueur a changé d'avis, il est encore temps de ne pas lancer la partie. Une fois le niveau sélectionné, c'est trop tard : la partie est insérée dans la base de données et la prochaine page ('presentationPartie.html') ne contient plus de menu.

- presentationPartie.html :

Si le joueur sélectionne un niveau auquel il a accès et qu'il n'est pas bloqué, il se retrouve sur cette page.

Elle permet de présenter au joueur des informations :

- sur le niveau : temps qu'il possède pour réaliser la figure et temps d'affichage de l'aide

- sur la figure : nombre de cases à colorier et nombre de couleurs différentes à utiliser

Un bouton HTML est présent pour diriger l'utilisateur vers la page 'partie.html'. On le prévient que dès qu'il clique sur le bouton, la partie commence et le chronomètre démarre.

- partie.html :

Grâce à l'identifiant de la figure stocké dans le dictionnaire « session », le code Python, permettant de retourner le template de la page « partie.html », interroge la base de données afin d'obtenir les informations suivantes : la liste des couleurs pour chaque case (qui permet de créer la grille « Représentation », c'est-à-dire la figure à réaliser), le niveau de la partie, la durée d'affichage et la durée de réalisation. De plus, il crée un code HTML qui constituera la grille où le joueur pourra réaliser sa figure. Une fois ces objets créés, on retourne la page.

Dès qu'on arrive sur cette page, le code JavaScript permet de cacher la grille et d'afficher seulement la représentation. Cela dure quinze secondes pour n'importe quel niveau. Ensuite, une fois ce laps de temps écoulé, c'est l'inverse : la représentation est cachée et c'est la grille vide qui apparaît. Dans le même temps, on envoie une requête afin d'obtenir le temps de réalisation et le temps d'affichage de l'aide (comme on vient de voir, ces variables ont été récupérées dans le code Python depuis la base). Le temps d'affichage permettra de connaitre la durée d'affichage de l'aide. Le temps de réalisation permet d'afficher dans un cadre noir le chronomètre indiquant le temps restant avant la fin de la partie.

Tout cela est résumé à travers le schéma suivant :

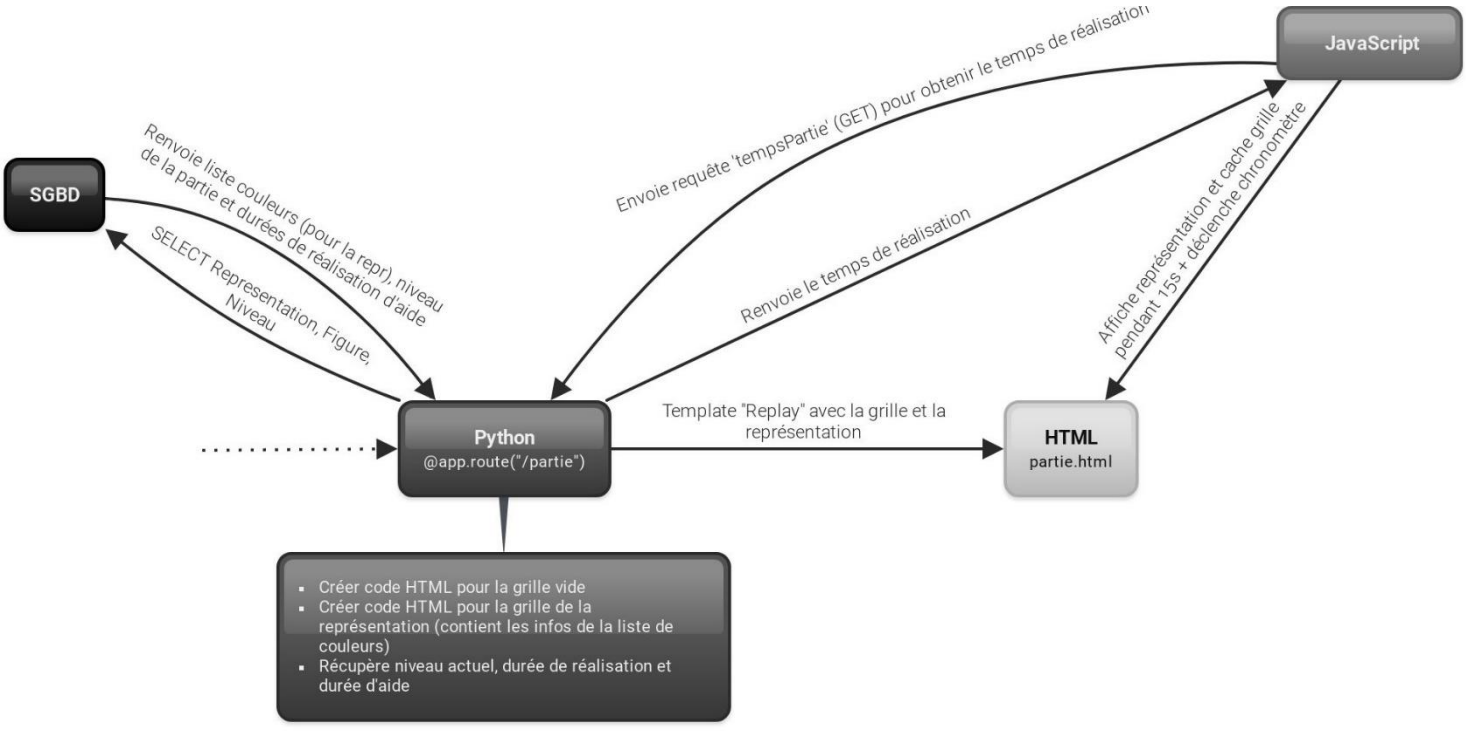


Figure 7 - Schéma illustrant les fonctionnalités mis en place dès que le joueur arrive sur la page "partie.html"

Une fois les quinze secondes écoulées, le joueur se retrouve donc devant une grille vide. Il possède à sa disposition six boutons colorés (bleu, rouge, orange, rose, violet et vert).

Ces boutons permettent tout simplement de choisir la couleur avec laquelle il souhaite colorier les cases. Pour connaître la couleur actuelle, nous disposons d'une variable globale dans le code JavaScript, par défaut cette couleur est bleue. Comme le montre le dessin ci-dessous, pour pouvoir la changer, le joueur doit cliquer sur un de ces boutons (ayant la classe « button_color »). Dans le JavaScript, lorsque cet événement se produit, on donne comme valeur à la variable globale l'identifiant du bouton cliqué (cet identifiant est une couleur).

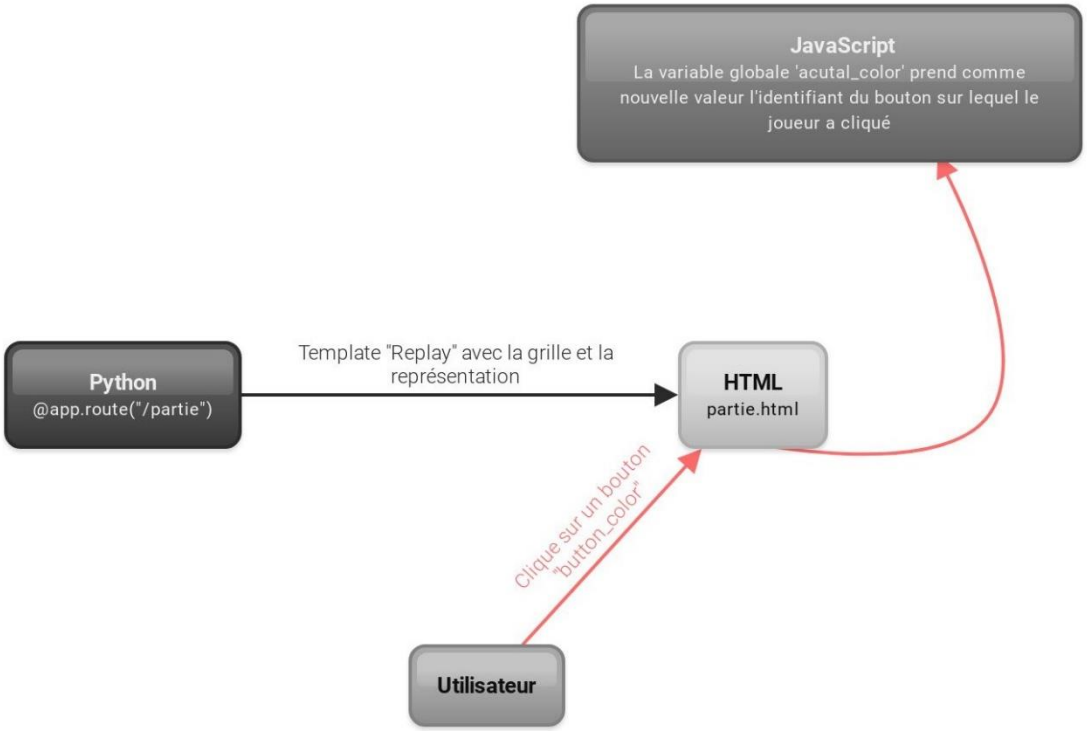


Figure 8 - Schéma illustrant le changement de couleur

Une fois que le joueur a sélectionné la couleur souhaitée, il peut alors colorier les cases de la grille.

Ces cases sont identifiées par la classe HTML « grid ». Dès lors qu'il clique sur une de ces cases, un code JavaScript se met en route. Ce code utilise la valeur de la variable globale 'actual_color' vu juste au-dessus. Deux cas possibles :

* Si la valeur de la couleur actuelle est la même que celle de la case que le joueur souhaite colorier (c'est-à-dire la même que la classe HTML de la case), alors on doit décolorer cette case. Pour cela, on doit modifier la classe en la mettant à « rien ».

* Si la valeur de la couleur actuelle est différente que celle de la case, alors on modifie la classe de cette case en lui attribuant la valeur de la variable 'actual_color'.

Cela est illustré par le schéma ci-dessous :

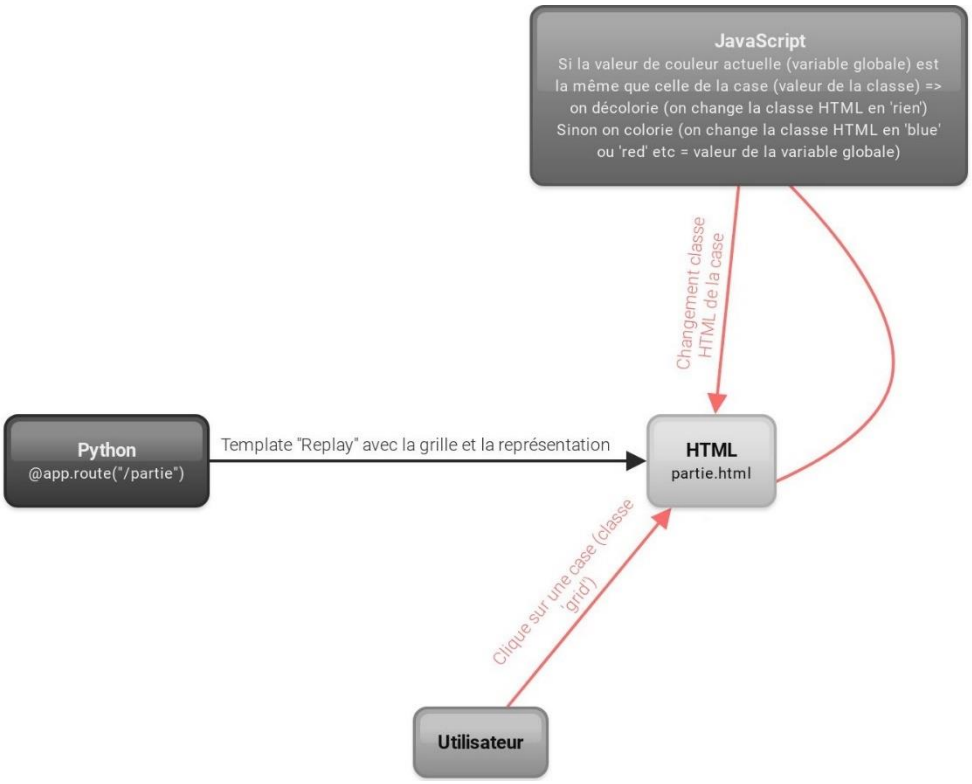


Figure 9 - Schéma illustrant la coloration (ou décoloration) d'une case

Par ailleurs, on doit également rajouter dans la table Historique chaque action réalisée par le joueur au cours d'une partie.

Dès que le joueur clique sur un objet HTML de classe « historique » (que ça soit un bouton ou une case), JavaScript permet de récupérer le type d'action et les coordonnées x et y pour les envoyer dans une requête. On récupère ces informations dans le code Python qui se chargera de les envoyer dans la base via la requête « inserer_historique ». Nous serons informés d'une erreur éventuelle grâce au code de retour de la procédure.

Afin de mieux comprendre cette fonctionnalité le schéma ci-dessous est utile.

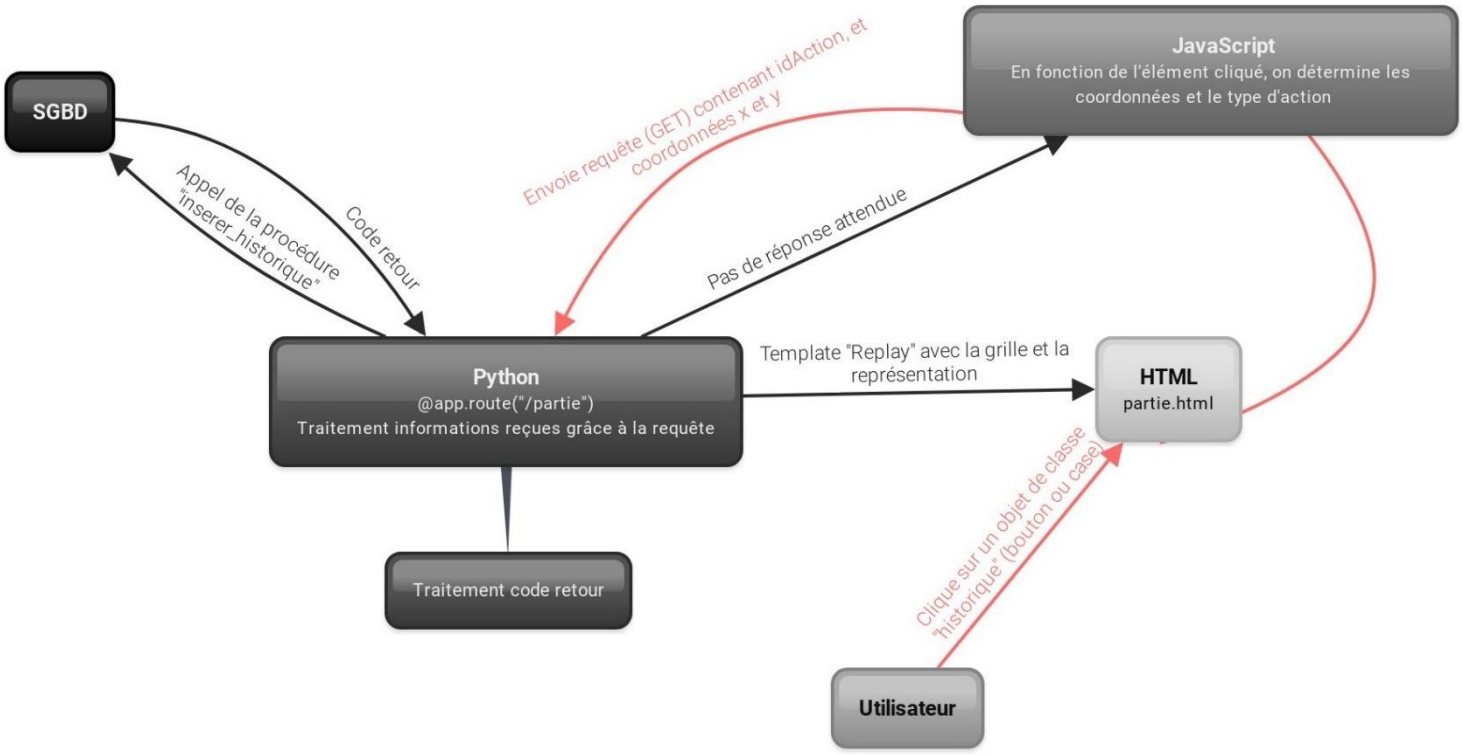


Figure 10 - Schéma montrant l'insertion des actions dans la table Historique

En plus des six boutons de couleur, le joueur possède un bouton « Aide » qui lui est utile lorsqu'il ne se souvient plus exactement de la figure à réaliser.

Lorsqu'il clique sur ce bouton, le code JavaScript permet de cacher la grille et d'afficher la représentation.

La durée dépend bien évidemment de la valeur de la variable 'dureeAffichage'.

Une fois le temps écoulé, on re-cache la représentation pour remettre la grille.

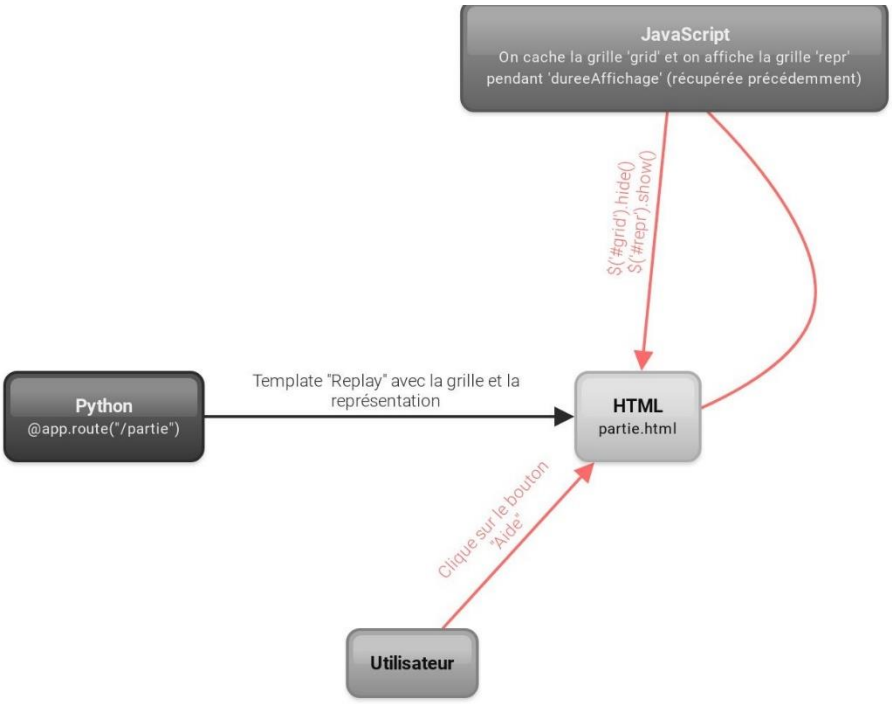


Figure 11 - Schéma illustrant la demande d'aide

Enfin, le dernier bouton mis à la disposition du joueur est le bouton « Validation » qui permet au joueur de soumettre sa figure à vérification lorsqu'il pense avoir réussi à reproduire l'originale.

Dès que le joueur clique sur ce bouton, on récupère dans le JavaScript les classes de chaque case de la grille et de la représentation que l'on stocke dans deux listes. On transforme ensuite ces listes en deux chaînes de caractères. Ces chaînes ne sont pas directement comparables car il existe des différences entre les classes d'une case « grid » et celles d'une case « repr ». On nettoie donc ces deux chaînes afin de pouvoir les comparer. Le joueur a gagné si et seulement si elles sont identiques.

L'étape suivante consiste à envoyer une requête dont les valeurs des paramètres (Temps, Date, Etat) varient en fonction du résultat de la comparaison.

On traite ces informations reçues dans le code Python, si le paramètre 'etat' vaut 'succes', on transforme les valeurs pour pouvoir les insérer dans la base de données (exemple : on met la date dans le format attendue), on appelle la procédure « fin_partie » et on retourne « finSucces ».

En revanche, si 'etat' vaut 'echec' on ne fait rien et on retourne 'perdu'.

En fonction de la réponse renvoyée par Python, JavaScript redirige vers la page « finPartieSucces.html » (si la réponse est « finSucces ») ou affiche un message d'erreur indiquant que la figure est incorrecte dans la page « partie.html» (si la réponse vaut « perdu »).

Il est important de signaler, qu'une requête contenant les mêmes paramètres est envoyée lorsque le compte à rebours est fini. Dans ce cas, le joueur a perdu, le code Python permet de traiter les paramètres reçus puis d'appeler la procédure « fin_partie ». Il retourne « finEchec » ce qui permet dans le code JavaScript de rediriger le joueur vers la page « finPartieEchec ». Avant la redirection on affiche un message au joueur pour lui indiquer que le temps est écoulé et donc qu'il a perdu grâce à la fonctionnalité HTML « alert box ».

Tout cela est une nouvelle fois résumé à l'aide d'un schéma faisant intervenir les différentes technologies utilisées pour mettre en place cette fonctionnalité.

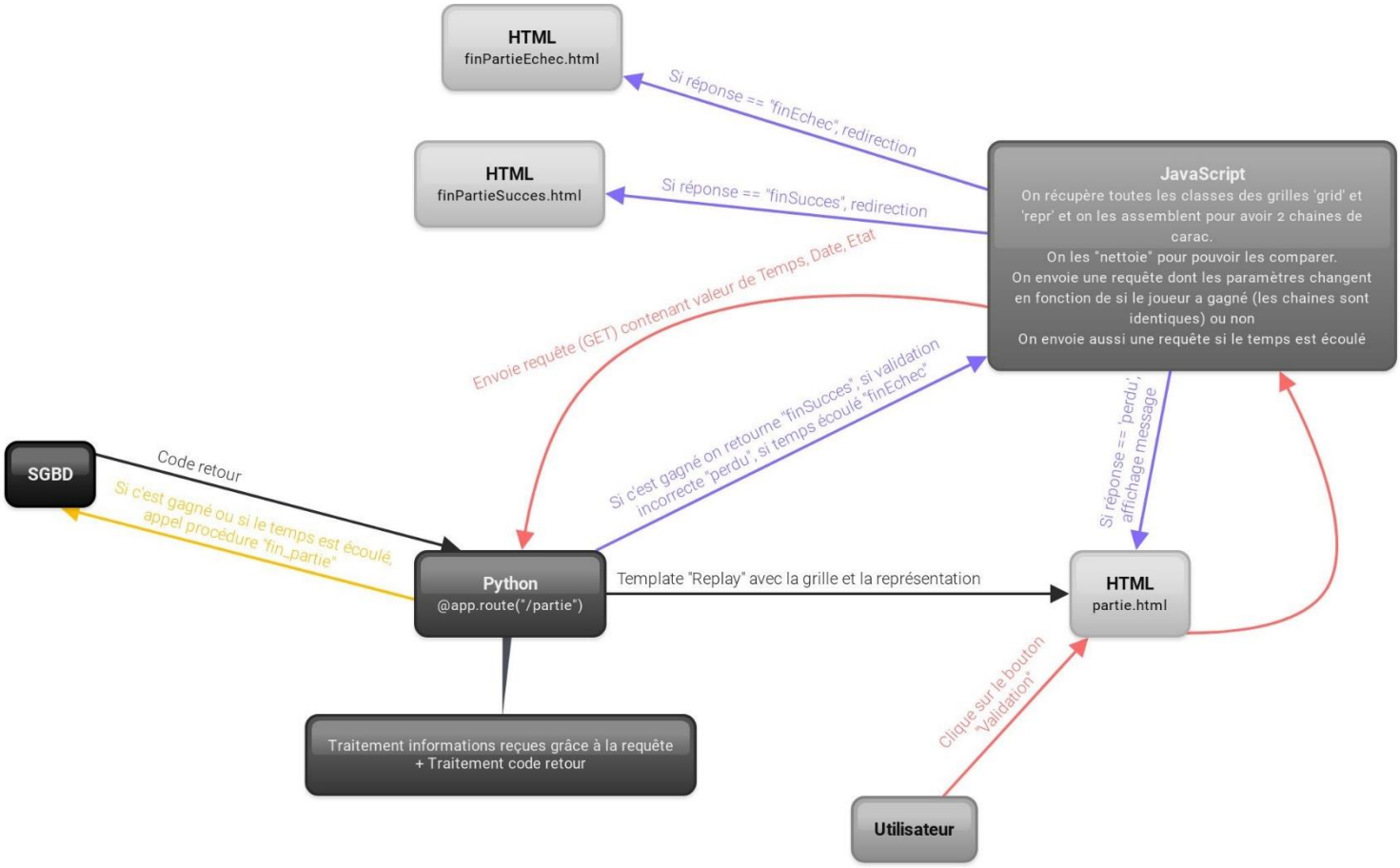


Figure 12 - Schéma illustrant les différentes étapes qui interviennent lors d'une validation

Remarque :

Lorsqu'un joueur réactualise la page, la partie recommence à zéro. Certes sa figure disparait et il doit recommencer mais comme le chronomètre recommence à zéro, on considère cela comme de la triche. Nous avons donc pris en compte cette pratique dans notre code JavaScript. Il permet, dès lors que cette situation arrive, de mettre fin à la partie et de rediriger le joueur vers la page « finPartieQuit.html ».

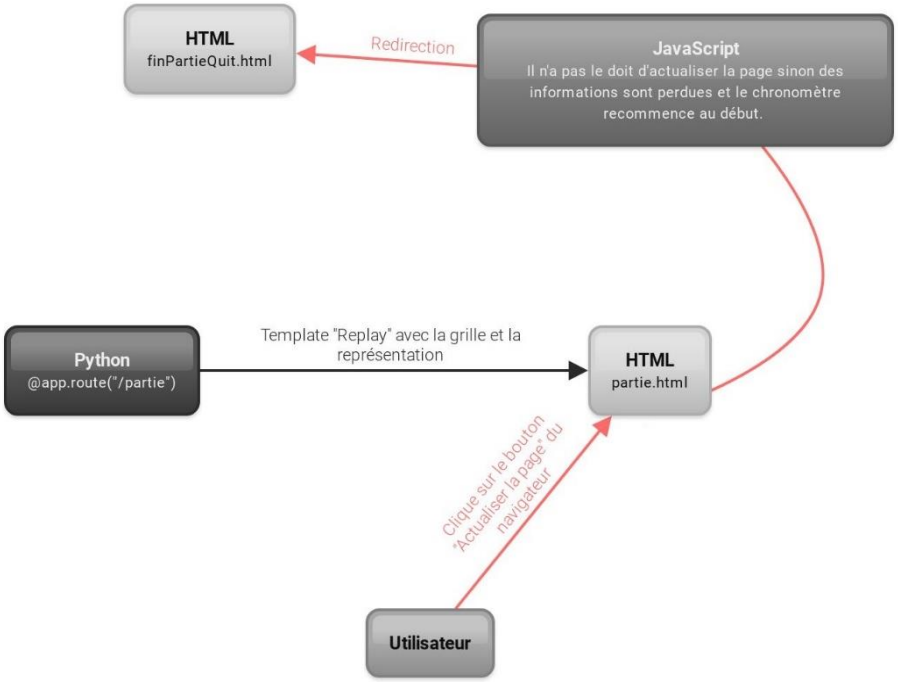


Figure 13 - Schéma illustrant la redirection suite à la réactualisation de la page "partie"

- finPartieQuit.html :

Avant de retourner le tempate de cette page, le code Python appelle la procédure « fin_partie » afin de mettre la partie à 'echec' dans la base de données.

Une fois que cette page est retournée, elle permet d'afficher un message indiquant au joueur que ce qu'il a fait est interdit et que la partie est perdue.

Remarque : Cette page a été rajoutée tardivement, elle n'apparait donc pas dans le schéma général.

- finPartieSucces.html :

Cette page indique la fin de la partie lorsque le joueur a gagné.

On y fait apparaître le score et un message de félicitation.

En fonction du résultat et du niveau, le message est différent :

- si le joueur a fait une partie de niveau 2 alors qu'il a déjà débloqué le niveau 3, on lui indique qu'il peut tenter des figures plus compliquées.

- si le joueur vient de réussir la deuxième figure d'un niveau (il avait déjà réalisé la première), il débloque donc le niveau supérieur, on le lui indique.

- si le joueur vient de réussir sa première figure d'un niveau, on lui indique qu'il ne lui reste plus qu'une figure pour débloquer le suivant.

- si le joueur a réussi mais il a un score de zéro (trop d'aide et/ou de validation incorrecte), on le prévient que malheureusement sa partie est comptabilisée comme perdue.

- enfin, si le joueur a déjà débloqué tous les niveaux, on lui affiche un message adéquat.

Pour finir, comme on peut le voir sur le schéma fonctionnel, cette page contient quatre boutons permettant soit de revenir à l'accueil, soit de rejouer, soit de revoir une partie ou soit de voir le tableau des HighScores.

- finPartieEchec.html :

Cette page est similaire à 'finPartieSucces.html' sauf qu'il n'y a pas l'affichage du score et que le message est un message d'encouragement et non de félicitation.

**** Revoir une partie*** **:**

Comme on peut le voir sur le schéma fonctionnel, lorsqu'on souhaite revoir une partie, la première page est 'choixPartie.html'.

- choixPartie.html :

Cette page permet à l'utilisateur de sélectionner, parmi toutes les parties qu'il a jouées, celle qu'il souhaite revoir. La page possède donc un formulaire sous forme de liste déroulante.

Avant de créer ce formulaire, nous devons récupérer la liste des parties et leurs informations depuis la base de données. Pour aider au mieux le joueur à sélectionner une partie précise, cette liste déroulante est très détaillée. Elle renseigne en effet le numéro de la partie, le numéro de la figure, le niveau, la date à laquelle la partie a été joué et enfin si elle a été réussie ou non. Ensuite, le code Python génère un code HTML pour créer le formulaire en y intégrant ces informations.

La page HTML est alors affichée et le joueur peut sélectionner la partie qu'il souhaite. Lorsqu'il valide son choix, une requête est envoyée, on récupère les informations envoyées (idPartie et idFigure) via Python afin de pouvoir les stocker dans le dictionnaire « session ». Cela nous permet également de récupérer des informations telles que le nombre de coups effectués lors de la partie, le score et le temps en interrogeant une nouvelle fois la base. Elles seront affichées dans la page suivante (« presentationReplay »).

A noter que sur le schéma ci-dessous, il y a deux cases « Python choixPartie ». Ces deux cases représentent la même fonction mais pour plus de clarté et pour avoir un ordre chronologique visuel, nous avons fait le choix d'en faire deux.

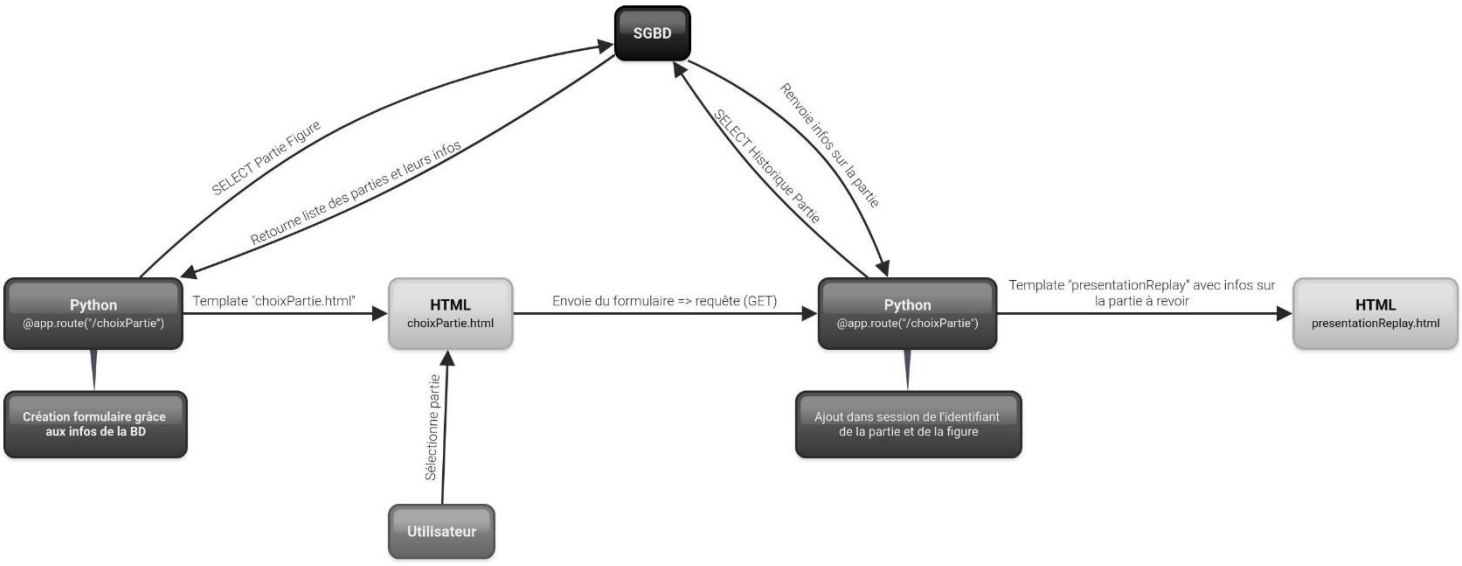


Figure 14 - Schéma montrant le choix de la partie avant un Replay

- presentationReplay.html :

Cette page est analogue à la page 'presentationPartie.html' que l'on a vue au-dessus. Elle permet d'informer le joueur sur le temps qu'il a mis pour faire cette partie, le score obtenu et le nombre de coups.

On l'informe également de la vitesse de replay : un coup toutes les deux secondes.

Pour finir, il y a un bouton permettant de se rendre à la page du replay.

- replay.html :

Avant que la page ne soit créée, le code Python interroge la base pour obtenir la liste des couleurs de chaque case qui permet de créer la grille « représentation » contenant la figure qu'a dû réaliser le joueur. On récupère aussi des listes pour les actions (liste des identifiants d'actions, liste des couleurs et liste des coordonnées x et y), ces informations sont ordonnées selon la valeur de l'attribut 'ordre'. Grâce à la longueur de ces listes, on peut définir une variable 'stop' qui nous permettra d'arrêter le replay.

Une fois que ces informations sont récupérées et que la représentation et la grille où l'on verra le replay sont créées, on peut créer la page en retournant son template.

Lors de l'arrivée sur la page, la représentation est affichée et la grille est cachée pendant 10s. Une fois ce laps de temps écoulé, c'est la grille qui apparaît et la représentation qui est cachée.

L'utilisateur doit alors appuyer sur le bouton 'Play' pour déclencher le replay. Lorsqu'il appuie, le code JavaScript permet de désactiver ce bouton, de créer la variable cpt=0 et de déclencher la fonction « play ». Cette dernière a pour objectif d'appeler la fonction « action » puis d'incrémenter le compteur 'cpt'. Elle commence donc quand le joueur clique sur le bouton « Play » puis elle se répète toutes les deux secondes.

La fonction « action », envoie une requête avec comme paramètre la valeur du compteur. On récupère cette valeur dans le code Python, Elle permet de récupérer les éléments des listes actions, couleurs et coordonnées ayant l'index 'cpt'. Si 'cpt' n'est pas égal à 'stop' (variable qu'on a vu juste au-dessus) alors la réponse à la requête correspond aux éléments des listes, sinon la réponse est « finReplay ».

Pour finir, si la réponse est différente de « finReplay », il reste donc des actions à afficher, le JavaScript permet alors de :

- soit modifier la classe HTML de la case en question si l'action est du coloriage (ou décoloriage)

- soit afficher un message si l'action est une demande d'aide ou une validation incorrecte.

En revanche, si la réponse est « finReplay », cela annonce la fin, l'utilisateur est automatiquement redirigé vers la page « finReplay.html ».

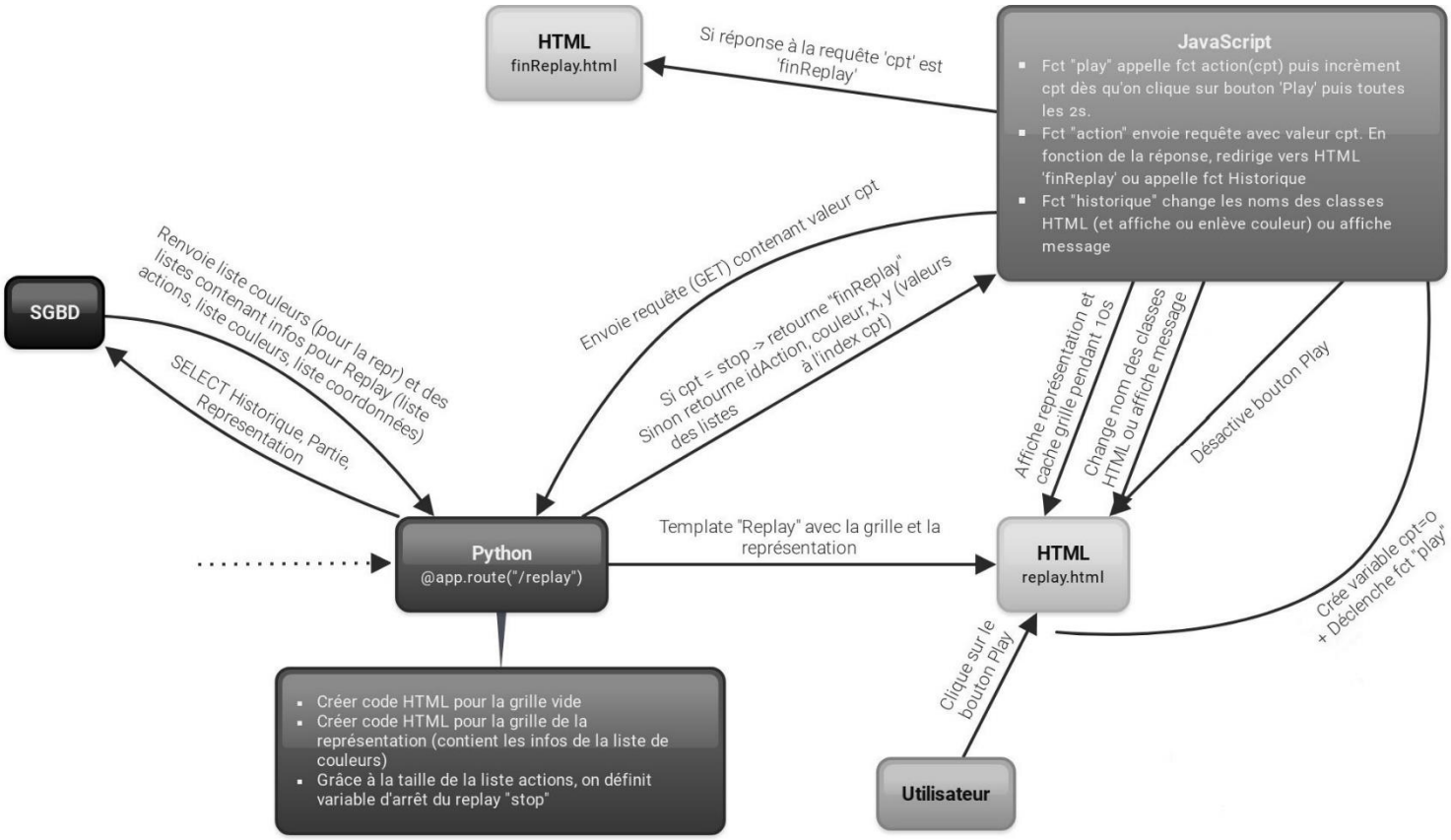


Figure 15 - Schéma illustrant le fonctionnement du Replay

<u>- finReplay.html :</u>

Cette page permet juste d'indiquer la fin du visionnage de la partie.

Elle contient également quatre boutons permettant soit de revenir à l'accueil, soit de rejouer, soit de revoir une partie ou soit de voir le tableau des HighScores.

2.3 Structure pages HTML

Pour faciliter la mise en page du site grâce au langage CSS mais aussi pour permettre une continuité entre les pages de même catégorie, nous avons construit nos pages avec une certaine logique. Le but de cette partie est donc de détailler cela.

Tout d'abord il est important de noter que nous avons adopté une convention pour les balises qui ne possèdent pas de balises fermantes (exemple les balises <img>). Cette convention est très simple, à la fin de ces balises on rajoute un slash. Par exemple *: *. Cette convention est utilisée par une grande partie des développeurs web, d'où notre souhait de l'adopter.

Avant de regarder en détail la structure (balises, identifiants, classes, etc) des pages pour chaque catégorie, il est important d'indiquer la présence d'éléments communs à toutes les pages. Ces éléments communs sont des divisions (balises) ayant un identifiant ou une classe permettant de mettre en relation les différentes sections (on verra par la suite que le contenu de nos pages est découpé en différentes sections), en-têtes, menus qui ont le même rôle. Cela permet de pouvoir les gérer de la même manière dans le fichier CSS.

Pour commencer, voyons les balises qui permettent dans le fichier CSS de structurer la page, c'est-à-dire de travailler la mise en page.

< div id = « wrapper »> comme son nom l'indique cette balise permet de regrouper (ou « envelopper ») tout ce qui appartient à la page hormis l'en-tête principal (voir un peu plus loin à quoi elle correspond). On verra qu'elle regroupera principalement les divisions « sidebar » et « main ».

< div id = « sidebar »> cette division permet de regrouper toutes les balises qui constituent la barre latérale (contenant notamment le menu).

< div id = « main »> cette division permet de regrouper tout ce qui est unique dans une page. En effet, le contenu de cette division est spécifique à la page en cours (contrairement à l'en-tête principal ou au menu qui eux sont identiques pour chaque page).

<div id = « banner »> cette division n'est présente que sur certaines pages. Elle permet de regrouper des éléments pour former une bannière contenant le titre de la page et quelques informations sur son contenu. Cette bannière, lorsqu'elle est présente, est située en haut de page.

Remarque : on utilise des identifiants, ces éléments sont uniques dans une page.

< div class = « inner »> référence de tout ce qui est à l'intérieur et plus précisément ce qui est à l'intérieur de la barre latérale (« sidebar ») ou du contenu unique de la page (« main »). Elle est particulièrement utile dans le CSS pour aligner les blocs, définir des marges, définir des bordures, etc. A noter qu'il y a également la présence de « inner2 » pour les pages sans « sidebar ».

<div class = « features »> a pour objectif de regrouper plusieurs articles (paragraphes ayant un titre) au sein d'une section (on rappelle que le contenu de nos pages est découpé en plusieurs sections). Cela permet avec le CSS de les aligner comme souhaité (deux par deux).

< div class = « content »> elle a le même rôle que « inner », la seule différence est qu'elle référence des éléments contenus à l'intérieur d'une bannière (<div id = « banner »>) ou d'un article (eux-mêmes contenus dans les divisions <div class = « features »>).

Remarque : on utilise ici des classes, ces éléments peuvent donc se retrouver plusieurs fois dans une page.

On va retrouver ces divisions tout au long de notre description catégorie par catégorie, c'est pour cela qu'il était important de faire une première approche.

Il existe bien évidemment d'autres identifiants et classes permettant de regrouper certains éléments HTML. Ces éléments sont présents plus ponctuellement (seulement sur certaines pages), on ne peut donc pas les détailler autant que les précédents. Voici la liste des principaux regroupements :

<header class = "major"> concerne les titres des sections (Il existe aussi « major2 »).

<p class="copyright"> concerne les droits d'auteurs situés dans le pied de page du « sidebar ».

 indique le nom d'une partie qui regroupe plusieurs liens dans le menu.

 regroupe les messages d'erreurs (permet notamment de les afficher en rouge grâce au CSS)

 référence les images servant d'illustration dans certains paragraphes.

 référence les images servant d'illustration dans les bannières.

 permet de centrer les sections (la grille, les boutons, etc) lorsque le joueur joue ou revoit une partie.

<table class="highscore"> , <th class="tab_bottom"> et <th class="tab_top"> permettent de gérer l'affichage des tableaux de Highscores.

Pour accéder à des éléments encore plus spécifiques dans le code JavaScript, il y a également la présence d'autres identifiants et/ou classes. Par exemple, l'identifiant à un bouton, à une case de la grille, etc. Nous ne les détaillerons pas ici.

Rentrons désormais un peu plus dans le détail en analysant les structures HTML en fonction de la catégorie.

2.3.1 Pages de la catégorie « connexion »

Comme nous l'avons vu précédemment, cette catégorie est constituée de trois pages ('index.html', 'connexion.html' et 'nvjoueur.html') qui ne contiennent pas de menu.

La structure de la page est donc la suivante :

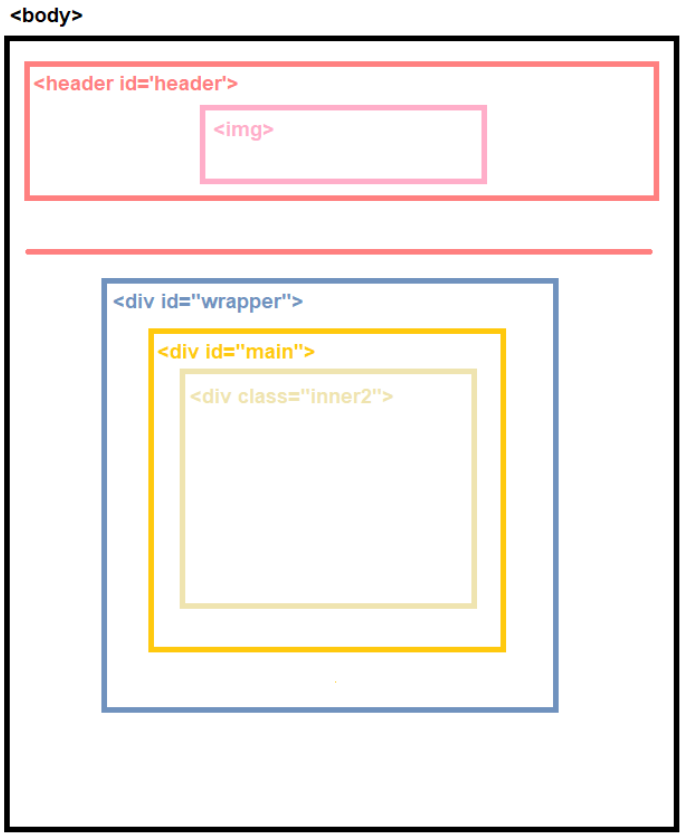


Figure 16 – Structure HTML des pages « connexion »

Tout d'abord, les pages de type « connexion » sont toutes constituées d'une balise « header ». Cette balise nous permet d'afficher en haut de chaque page le logo et le nom du jeu (contenus dans la balise « img »). On retrouvera d'ailleurs cet en-tête sur toutes les pages du site.

Ensuite, comme évoqué dans l'introduction de cette partie, on retrouve les balises « wrapper », « main » et « inner2 » qui permettent de faire la mise en page grâce au fichier CSS.

Pour finir, c'est donc à l'intérieur de la balise qu'on aura des contenus différents selon les pages. On y trouvera généralement des balises <section> composées d'en-tête (<hearder>), de titre (<h2>), de boutons (<button>), de textes (<p>), etc.

Les pages « connexion.html » et « nvjoueur.html » contiennent un formulaire (balise <form>) permettant d'envoyer le pseudo et le mot de passe.

Passons maintenant aux pages ayant une barre latérale (« sidebar »).

2.3.2 Pages de la catégorie « Menu »

Pour rappel, cette catégorie est constituée de six pages ('accueil.html', 'regles.html', 'contact.html', 'niveau.html', 'choixPartie.html' et 'highscore.htlm'). Elles possèdent donc une barre latérale contenant un menu permettant de naviguer entre ces différentes pages. Toutes ces pages possèdent la même structure :

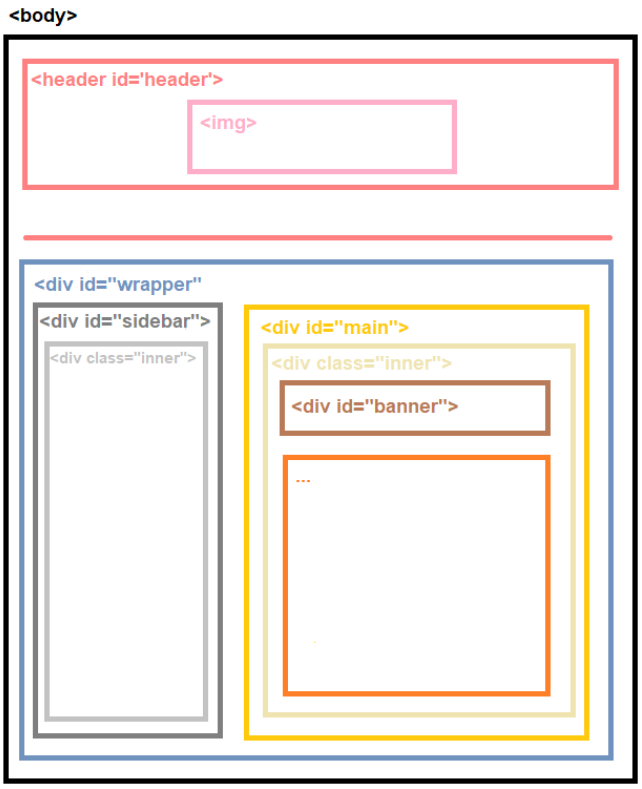


Figure 17– Structure HTML des pages « menu »

On peut tout d'abord remarquer qu'il y a toujours l'en-tête général (<header id= « header »>).

Ensuite, la division « wrapper » enveloppe les divisions « main » et « sidebar » comme vu auparavant. De plus, ces deux divisions contiennent bien une division « inner ».

En ce qui concerne la partie « main », on remarque que toutes les pages contiennent une bannière. Elle permet de présenter aux joueurs les différentes informations contenues dans la page. Comme évoquée précédemment, à l'intérieur de cette balise il y a la présence de la balise <div class = « content »>. De plus, ces bannières contiennent les balises <header>, <h2> et <p>.

Ensuite, la partie « main » contient tout ce qui est spécifique à une page (rectangle orange). On ne va pas rentrer dans le détail page par page pour savoir ce qu'elles contiennent. Il est juste important de savoir que c'est une nouvelle fois un ensemble de sections. Ces sections contiennent généralement des paragraphes (<p>), des boutons (<button>), des en-têtes (<header>), des titres (<h1>, <h2>, <h3>, <h4>), des articles (<article>), des listes (<ul>), etc.

Pour finir, voyons plus en détail de quoi est constituée la barre latérale (elle est identique pour toutes les pages de cette catégorie) :

La barre latérale est constituée de trois partie :

* <nav id=menu »> : Comme son nom l'indique cette partie correspond au menu. Elle est constituée d'un en-tête pour afficher un titre et d'une liste contenant les différents liens.

* <section> : Contient des informations sur le site.

* <footer id= « footer »> : Correspond au pied de page. Il permet de spécifier les droits d'auteurs.

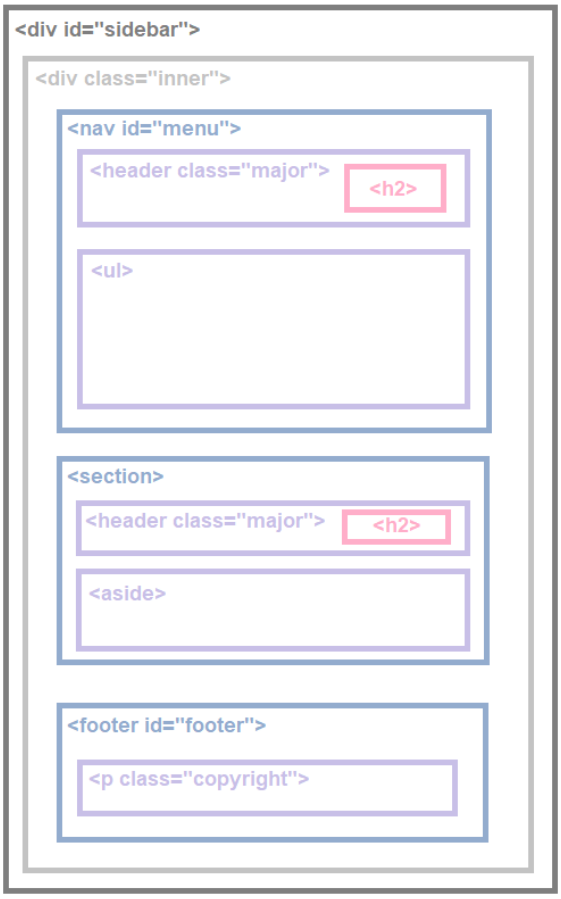


Figure 18 - Détail de la barre latérale pour la catégorie "Menu"

Pour finir, il nous reste à étudier la structure des pages de la catégorie « dynamique ».

2.3.3 Pages de la catégorie « dynamique »

Nous avons vu auparavant, cette catégorie est constituée de sept pages ('niveau.html', 'choixPartie.html', 'partie.html', 'replay.html', 'finReplay.html', 'finPartieSucces.htlm' et 'finPartieEchec.htlm'). Ce sont les pages « dynamiques » (JavaScript) qui permettent de jouer une partie ou de voir un replay. Elles possèdent une barre latérale, mais la différence avec les pages de la catégorie « Menu » c'est que cette barre latérale ne possède pas de menu pour empêcher le joueur de naviguer. Toutes ces pages possèdent la même structure :

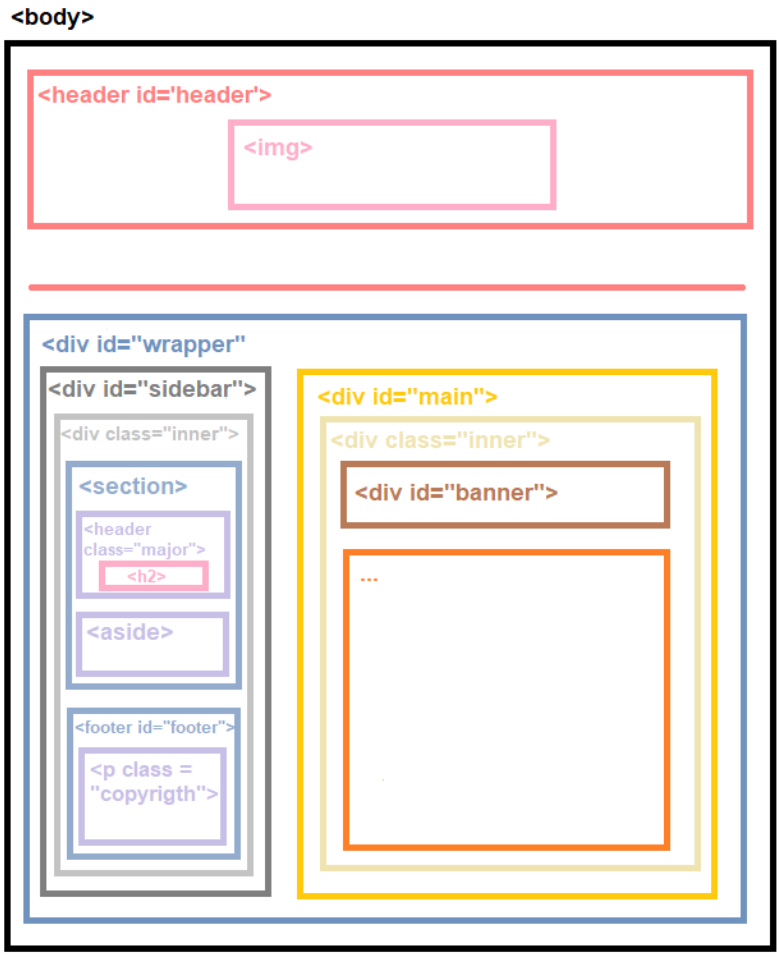


Figure 19- Structure HTML des pages de la catégorie "dynamique"

La structure HTML est, comme on peut le voir, exactement la même que la catégorie « Menu », la seule différence se trouve donc au niveau de la barre latérale (« sidebar ») qui possède seulement une section et un pied de page mais pas de menu.

Remarque : Seules les pages « finReplay », « finPartieSucces.htlm » et « finPartieEchec.htlm » ne possèdent pas de bannière (rectangle marron).

Comme précédemment, la partie « main » contient tout ce qui est spécifique à une page (rectangle orange). On ne rentrera pas en détail pour chaque page. Il est juste bon de savoir que ces pages contiennent généralement des paragraphes (<p>), des grilles (<table>), des boutons (<button>), des en-têtes (<header>), des titres (<h1>, <h2>, <h3>, <h4>), des articles (<article>), etc.

Nous vous avons donc présenté ici les grandes lignes des structures HTML de nos pages en essayant d'être le plus clair possible. Bien sûr, pour rentrer plus en détail, il est plus utile de regarder directement le code.

Pour conclure, on peut indiquer que toutes les pages HTML et le fichier CSS sont valides au sens W3C.

2.4 Choix

2.4.1 Niveaux

Notre version du Pixel Factory possède quatre niveaux qui sont eux-mêmes constitués de deux figures.

- difficulté :

Pour différencier les échelons de difficulté des niveaux, on a opté de jouer sur plusieurs aspects : le temps d'affichage de la figure, le temps que possède le joueur pour réaliser la figure, le nombre de couleurs différentes d'une figure et le nombre de cases à colorier. Ce qui donne :

- niveau 1 : temps d'aide 15 secondes / temps de réalisation 180 secondes
- niveau 2 : temps d'aide 10 secondes / temps de réalisation 160 secondes
- niveau 3 : temps d'aide 15 secondes / temps de réalisation 140 secondes
- niveau 4 : temps d'aide 20 secondes / temps de réalisation 120 secondes

Il peut paraitre incohérent d'augmenter de temps de l'aide alors que le niveau devient plus dur, mais cela s'explique par le fait que les figures deviennent de plus en plus complexes. Elles possèdent donc plus de couleurs différentes et plus de cases à colorier, le temps d'affichage doit donc être un peu plus élevé. Par ailleurs, le temps de réalisation lui diminue de plus en plus.

Remarque : Quel que soit le niveau, le temps d'affichage en début de partie est de quinze secondes.

- déblocage d'un niveau :

Pour pouvoir débloquer un niveau supérieur, le joueur doit obligatoirement réaliser les deux figures du niveau. De plus, on considère qu'une partie est réussie seulement si le score est supérieur à zéro (voir la partie 2.4.3 Score pour plus de détail).

2.4.2 Lors d'une partie

- sélection d'une figure :

Comme nous l'avons évoqué antérieurement, il y a quatre niveaux qui sont constitués de deux figures. Lors du début d'une partie, plutôt que d'attribuer aléatoirement une des deux figures du niveau souhaité par le joueur, nous avons fait un choix un peu plus complexe. Pour cela, nous devons distinguer plusieurs cas :

- le joueur a réalisé une des deux figures, dans ce cas on attribue automatiquement la figure restante.

- le joueur n'a réalisé aucune figure ou a réalisé toutes les figures du niveau, dans ce cas on attribue aléatoirement une des figures.

A noter, on a fait en sorte que la fonction Python, qui attribue la figure de la partie, puisse le faire si le nombre de figure par niveau est supérieur à 2. Autrement dit, on peut rajouter autant de figures par niveau que l'on veut et notre site web fonctionnera toujours.

- affichage :

Pour corser le jeu, on a fait le choix de changer le fond entre la grille permettant de réaliser la figure et la grille permettant d'afficher la représentation de la vraie figure. Cela peut paraitre inutile, mais nous avons fait le test et le fait de changer le fond rend l'exercice beaucoup plus difficile. Le fond de l'aide est noir, celui de la réalisation est beige.

- couleur :

Toujours dans l'optique de corser le jeu, nous avons décidé de laisser dans le choix des couleurs celles qui ne sont pas nécessaire à la réalisation de la figure. Sachant qu'en plus parmi nos six couleurs choisies certaines se ressemblent fortement : le bleu et le violet ou pire encore le rouge et l'orange.

- quitter une partie :

Comme vue précédemment, si un joueur quitte une partie en fermant la page ou le site, on ne peut pas appeler la procédure « fin_partie » et définir la partie comme perdue. Par conséquent, lorsqu'il se reconnectera on utilisera une procédure qui mettra l'état de la partie 'NULL' à 'echec' (normalement il ne peut pas y en avoir plusieurs vu que cette procédure est appelée à chaque fois qu'il se connecte). Pour la date, on mettra la date actuelle, cela pénalisera le joueur puisqu'il aura une partie de plus perdue dans l'heure alors que peut-être que la partie qu'il a quitté en fermant le site datait d'il y a plusieurs heures ou plusieurs jours. Cette pénalité est voulue. On aurait pu très bien créer une variable « dateDernièreConnexion » et obtenir la vraie date de la partie, mais nous souhaitions vraiment sanctionner les joueurs qui quittent parce qu'ils ne réussissent pas leur partie. En revanche, si un joueur quitte par maladresse nous estimons qu'il se reconnectera dans la foulée donc il ne sera pénalisé que de quelques minutes (même s'il est vrai qu'il aurait peut-être réussi sa partie, nous sanctionnons donc sa maladresse en considérant la partie comme perdue).

2.4.3 Score

- calcul :

Pour pouvoir calculer le score d'une partie, nous nous basons sur trois éléments : le temps que le joueur a mis, le nombre de fois où il a demandé de l'aide et le nombre de validations incorrectes.

Pour rappel, la formule utilisée est :

*score = nombre d'aide * -100 + (nombre de validation-1) * -50 + temps * -3 + 1000*

Comme on calcule le score uniquement pour une partie réussie il faut enlever un au nombre de validations. En effet, la dernière validation est forcément une validation « juste » et non « incorrecte ».

- partie considérée comme réussie :

On a vu que pour qu'une partie soit considérée comme réussie, il faut que le score soit supérieur à zéro. Ainsi, si le joueur a reproduit la figure demandée à l'identique mais qu'il a utilisé trop de fois l'aide ou validé sa figure alors qu'elle est incorrecte, sa partie ne sera pas considérée comme gagnée.

2.4.4 Highscores

Concernant les Highscores, le sujet nous imposait de créer deux types de tableaux pour les joueurs : un par niveau de jeu et un pour la journée en cours.

Afin de permettre aux joueurs de se comparer aux autres, nous avons fait le choix de rajouter des informations. En effet, en plus d'indiquer le meilleur score pour le joueur connecté, on indique également le meilleur score pour tous joueurs confondus. On affiche également le pseudo du meilleur joueur. Les tableaux sont donc constitués de trois lignes : le score du joueur actuel, le score du meilleur joueur et son pseudo.

Nous venons de vous présenter les principaux choix que nous avons faits lors de la réalisation du site web, cependant certains que nous considérons mineurs n'ont pas été détaillés ici.

2.5 Problèmes rencontrés

Pour commencer, le premier problème que nous avons rencontré lorsqu'on a débuté la partie web, a été de savoir par quoi commencer. En effet, on avait l'impression qu'il y avait énormément de choses à faire et que nous serions incapables de tout faire dans le temps imparti. La solution adoptée était très simple, elle a consisté à établir un planning en mettant l'accent sur les points à réaliser en priorité et ceux qui étaient moins urgents ou facultatifs. On a alors commencé par dresser les structures HTML des pages puis programmer le code HTML correspondant. Ensuite, nous nous sommes attaqués à l'interaction BD/Web et au code Python. Rendre dynamiques les pages permettant de jouer ou de revoir une partie, grâce au langage JavaScript, a constitué la phase suivante. Pour finir, étant donné que ce point était facultatif, nous avons mis en page le site grâce au langage CSS. Ce choix a été plutôt payant puisque nous avons réussi à finir le projet à temps, bien que les deux semaines de délais supplémentaires nous aient été utiles pour finaliser certains points.

L'une des raisons au fait que nous étions un peu débordés lorsque nous sommes passés à la partie web était que la plupart des langages nous étaient inconnus. En effet c'était la première fois que nous devions programmer en HTML, en CSS, en JavaScript (et sa bibliothèque jQuery) et utiliser le Framework Flask. Il n'y a que le langage Python que nous manipulons depuis plusieurs années. Bien évidemment, nous avons reçu au cours de cette UE des enseignements sur tous ces langages (hormis le langage CSS qui était facultatif pour le projet). Mais comme nous avons aussi suivi des cours pour la base de données, les cours portant sur la programmation web étaient peu nombreux, le rythme était donc soutenu et beaucoup d'informations nous ont été transmises. Pour compléter ces enseignements, nous avons fait le choix de suivre des formations en ligne. Le site OpenClassRooms nous a été particulièrement utile. Ce choix nous a demandé beaucoup d'investissement que nous avons pris sur notre temps libre.

Pour finir, l'un des problèmes majeurs rencontré a été la modification de la base de données à plusieurs reprises suite à la constatation de quelques bugs lorsque nous avons commencé à coder la partie web. En effet, comme nous l'avions vu juste au-dessus, la majorité des langages nous étaient inconnus au moment de la conception de la base de données donc nous ne savions pas vraiment comment allaient être utilisés certains attributs de la base ou les procédures. Les changements dans la base de données sont donc intervenus assez souvent durant la programmation de la partie web. C'est Stévan qui s'en est chargé en tant qu'administrateur de la base.

Conclusion

Pour finir cette présentation de notre projet, on peut commencer par indiquer que l'objectif principal du projet, à savoir concevoir une site web s'appuyant sur une base de données a été atteint, la vidéo mettant en scène quatre « use cases » permet d'en témoigner. Par ailleurs, toutes les contraintes imposées par le sujet ont été respectées. Effectivement, le jeu en ligne que nous avons créé est programmé en Python/HTML-CSS/JavaScript et repose sur une base de données Oracle. Il permet le fonctionnement en mode mono-joueur, il comporte plusieurs niveaux de jeux, il met en place une limitation des parties, il permet l'affichage de tableaux de Highscores et pour finir les parties sont enregistrées sur la base de données ce qui permet à un joueur de revoir l'une de ses parties. Les traitements à prévoir côté SGBD, à savoir la création d'au moins trois procédures PL/SQL et d'au moins trois triggers ont également été effectués.

Cependant, certains points peuvent être améliorés afin d'avoir un site web encore plus proche des sites web professionnels. La plupart des améliorations que nous allons énumérer, n'ont pas été réalisées par manque de temps et non pas par manque de connaissance. En voici une liste non exhaustive :

- proposer un bouton permettant de quitter le replay en cours

- choisir la vitesse du replay (actuellement la vitesse est d'un coup toutes les deux secondes)

- possibilité pour un joueur de changer de mot de passe

- mettre en place un système de bonus pour le score quand un joueur réussit plusieurs figures d'affilées

- rajouter des classements (top 3, top 5 ou top 10) avec les Highscores, cela permettrait d'avoir un peu plus d'informations que seulement le 1er joueur

- déployer le site sur un serveur à distance et permettre que deux joueurs puissent jouer en simultané

- mettre un bouton « Abandonner » pour éviter qu'un joueur attende la fin de la partie ou du replay

Toutefois au cours de ce projet, nous avons fait preuve d'initiative en faisant une jolie mise en page du site grâce au langage CSS , en ajoutant des pages permettant de nous

contacter ou de connaître les règles du jeu, en donnant la possibilité de choisir un replay de n'importe quelle partie, en pénalisant un joueur qui abuse de l'aide ou qui valide trop de fois en se trompant et en nous rajoutant certaines contraintes comme nous l'avons vu dans la partie 2.4. C'est ce qui nous a permis d'avoir un site web encore plus performant et semblable à de « vrais » jeux en ligne.

Pour la suite de notre formation et même dans l'optique de notre vie professionnelle, le fait de revoir, de consolider et d'approfondir nos connaissances en base de données nous sera forcément utile. De plus, l'apprentissage de certains langages web peut constituer un plus sur le curriculum vitæ. Cela fut également enrichissant puisque nous utilisons tous les jours des sites internet sans forcément connaître toutes les technologies qui se cachent derrière. Même si nous n'avons bien sûr pas abordé toutes les notions, nous avons désormais une petite idée de ce qui permet de les créer. Par ailleurs, apprendre à s'autoformer sur des notions non développées en cours a été précieux puisque c'est une situation à laquelle on risque de devoir faire face au cours de notre carrière. Pour finir, réussir à mener, en groupe, un projet de bout en bout dans le but de résoudre un problème bien précis en faisant intervenir plusieurs domaines est bien évidemment une situation que l'on rencontrera souvent à travers notre futur métier.

Sources

Pour mener à bien ce projet, qui nécessitait d'apprendre beaucoup de notions, nous avons dû nous documenter. Voici les principales ressources qui nous ont aidés tout au long du projet.

Premièrement les cours de Madame PINEL-SAUVAGNAT Karen et Monsieur MORENO Jose reçus au cours des enseignements du seconde semestre de 3e année de licence de Statistique et Informatique Décisionnelle (SID) à l'université Paul Sabatier. Ces cours portaient sur les langages PL/SQL, HTML, Python et JavaScript. Les notions de jQuery et AJAX ont également été abordées. Les cours du premier semestre réalisés par Monsieur MORVAN Franck pour l'UE « Concepts Fondamentaux des Bases de Données » et Madame LECHAMINE-TAMINE Lynda pour l'UE « Langages de Requêtes » nous ont également été d'une aide précieuse.

En complément des cours reçus au cours de notre formation, nous avons approfondi l'apprentissage de ces nouveaux langages grâce au site OpenClassroom. Plus précisément nous avons suivi trois cours :

- Apprenez à créer votre site web avec HTML5 et CSS3 (Mathieu NEBRA)

- Apprenez à programmer avec JavaScript (Will ALEXANDER)

- Écrivez du JavaScript pour le web (Fabien HENON)

On peut également citer deux autres cours supplémentaires, nous ne les avons pas suivis à proprement parler, mais plutôt regardés partiellement :

- Simplifiez vos développements JavaScript avec jQuery (Michel MARTIN)

- Dynamisez vos sites web avec JavaScript ! (Johann PARDANAUD et Sébastien)

Ensuite, concernant les sites qui nous ont aidés, on peut citer :

- https://www.w3schools.com

- https://developer.mozilla.org

Pour finir, le code HTML et CSS a été inspiré du site : https://html5up.net (version Editorial).

Index