Université de Genève

Interaction Multimodale et Affective
D400002

Projet: Play Maths

Author: Sajaendra Thevamanoharan

E-mail: Sajaendra.Thevamanoharan@etu.unige.ch

Author: Deniz Sungurtekin

E-mail: Deniz.Sungurtekin@etu.unige.ch

Juin 2021

1 Introduction

L'objectif de ce projet est de créer une application permettant à de jeunes utilisateurs de s'exercer aux calculs mentaux de manière ludique. L'idée est de concevoir un jeu relaxant s'adaptant à certains signaux biologiques du joueur afin de trouver une évolution de la difficulté optimale pour permettre un meilleur apprentissage. Le système demandera donc à l'utilisateur de résoudre une série d'équations devenant de plus en plus complexe selon les signaux perçus. Dans ce rapport, nous discuterons donc du développement de l'application où nous décrirons les technologies utilisées et les difficultés rencontrées, puis nous analyserons les données obtenues après avoir fait essayer le jeu à deux groupes de quatre participants sur quatre parties.

Avant même de procéder à l'implémentation du système et à l'analyse des données, nous avons quelques hypothèses de recherche générales. Selon nous le stress a un impact important sur la performance du joueur et son apprentissage, nous supposons qu'il faut un niveau de stress suffisamment élevé pour stimuler l'utilisateur sans pour autant dépasser une certaine limite qui nuirait à sa capacité de raisonnement.
De plus, il serait intéressant de visualiser l'impact du système sur le joueur. Selon plusieurs études, le bon stress correspond à une dépense énergétique adaptée à l'action. Dans notre cas, l'action correspond donc à la résolution d'équations qui deviennent de plus en plus complexes de manière plus ou moins constante. Par conséquent, le jeu étant conçu pour avoir une ambiance apaisante et une difficulté initiale très simple, nous voudrions à premier abord observer un niveau de stress bas qui augmentera de manière constante jusqu'à la fin d'une partie.

Finalement, nous voulons étudier l'évolution des performances pour un même utilisateur sur plusieurs parties afin de savoir si l'utilisateur s'améliore. Idéalement, le joueur devrait avoir des meilleurs scores au fil des parties dues au fait de pratiquer et l'envie de "scorer". De plus, la moitié des participants utiliseront le Makey-Makey pour sélectionner leur réponse, nous essaierons de déduire si cela est bénéfique à leur performance. Étant donné que le processus de sélection par clavier est remplacé par le Makey Makey, nous présumons qu'il est plus intuitif d'appuyer sur des gros objets posés dans l'ordre des solutions proposées et donc que cela réduit potentiellement le temps de réponse.

2 Développement

2.1 Unity

La partie principale de l'application a été développée sous la version 2020.3.3f1 d'Unity. Le jeu est constitué de trois scènes, la première est une simple interface graphique composée de plusieurs éléments simples :

- Deux fenêtres de saisies où l'utilisateur indique son prénom et son âge.
- Un texte indiquant le titre de l'application.
- L'image de fond du jeu.
- Un bouton "Start" qui effectue la transition sur la deuxième scène et enregistre les informations entrées.

Voici un aperçu:

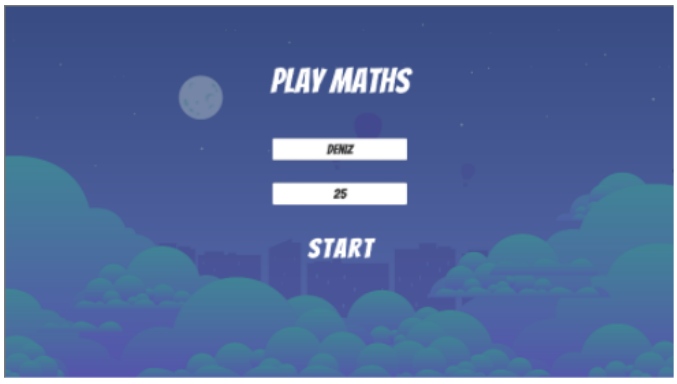

Figure 1: Écran Titre/Scène 1

Une fois le bouton "Start" appuyé le jeu se lance, cette scène représente la partie principale du jeu. C'est ici que le joueur verra défiler les équations et devra y trouver une réponse avant que celle-ci atteigne le bas de l'écran. Le défilement est donné par le mouvement de la caméra sur deux identiques images de fond possédant un texte représentant nos équations. Lorsque l'une de ces images sort du champ de vision elle se positionne au-dessus de l'image suivante, ainsi le joueur à l'illusion de monter. À cela s'ajoutent les objets qui monte avec la caméra mais qui sont en réalité immobiles. Ces éléments sont placés dans un canvas qui est toujours affiché à l'écran. Ce canvas est constitué de:

- Quatre boutons qui affichent les réponses possibles de l'équation visible.
- Quatre coeurs représentant la vie du joueur.
- Le score du joueur.
- Un bouton Mute/Unmute qui désactive/active le son du jeu.

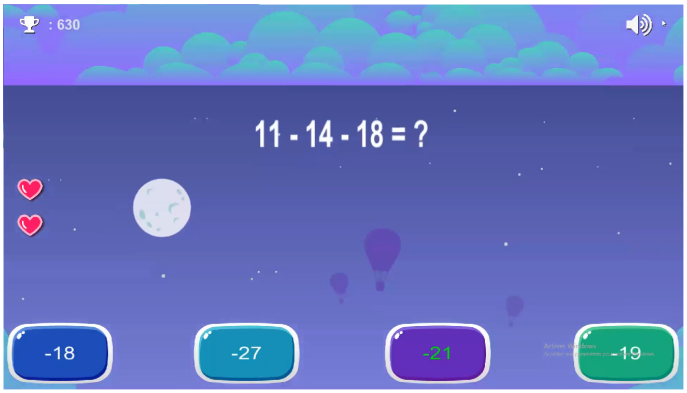

Figure 2: Scène 2

Au lancement de la scène, on génère une équation à deux termes dont l'opérateur (addition ou soustraction) et les nombres sont choisis aléatoirement pour les deux images. Une fois généré, on calcule et enregistre la réponse associée à chaque image puis on s'assure que celle-ci soit affichée sur un des boutons à une position aléatoire. En ce qui concerne les autres boutons, il est important de s'assurer que ceux-ci ne possèdent pas aussi la bonne réponse et qu'ils affichent tous des valeurs différentes. Comme expliqué précédemment, une fois qu'une image n'est plus visible dans le champ de vision celle-ci se positionne au-dessus de l'image suivante et c'est à ce moment-là qu'une nouvelle équation lui ai attribué.
Quant à la vérification des réponses, elle s'effectue lorsque l'équation dépasse une ligne horizontale invisible se trouvant au milieu de l'axe y des boutons. L'utilisateur peut sélectionner une des quatre propositions en appuyant sur la touche correspondante du clavier (les touches "1 2 3 4","A S D F", les flèches directionnelles ou le Makey-Makey). Si la réponse est bonne le joueur garde ses vies et augmente son score, sinon le joueur perd une vie et un certain nombre de points dépendant de l'état courant du jeu que nous détaillerons par la suite.

Pour concevoir un jeu qui puisse s'adapter à son utilisateur, il nous est nécessaire de définir une notion de difficulté qui englobe plusieurs caractéristiques du jeu. Nous avons décidé de considérer différents paramètres:

- La vitesse de défilement de la caméra.
- La grandeur absolue des termes.
- Le nombre de termes et d'opérateur dans une équation.
- La proximité des réponses proposées.

Ces paramètres sont mis à jour à chaque changement d'image selon la valeur de la difficulté courante initialisé à 1.0 qui est un nombre décimal allant jusqu'à 10.0. La mise à jour de la difficulté suit un processus qui inclut la captation de signaux biologiques du joueur que nous détaillerons dans la section "Python".

Pour ce qui est de la vitesse de défilement elle suit une formule précise dépendant de la difficulté:

$$MouvementSpeed = 1 + \frac{2}{9} * (difficulte\_courante - 1)$$

Cette formule est un simple mapping de l'intervalle [1,10] à [1,3] dont la borne maximale de la vitesse de défilement a été choisie de manière empirique en choisissant une vitesse maximale qui rend la résolution de l'équation difficile mais pas impossible.
Initialement, les équations possèdent deux termes entre 0 et 10 et un seul opérateur. L'intervalle dans lequel ces nombres sont choisis s'élargit à mesure que la difficulté croît puisque celui-ci est donné par:

$$[0, \lfloor 10 + difficulte\_courante \rfloor [$$

donc à la difficulté maximale un terme pourra au maximum avoir la valeur absolue de 19. De plus, à partir d'un niveau de difficulté de 7.5 les équations sont générées avec 3 termes et deux opérateurs. À ce moment-là, la vitesse de défilement redescend d'un cran et suis une autre formule de mis à jour que l'on décrira dans la section "Python". Ce changement a été fait afin d'éviter une augmentation trop brusque de la difficulté ressentie par l'utilisateur.
À propos de la mis à jour du score, elle dépend également de la difficulté. L'idée est de pénaliser l'utilisateur pour des erreurs simples et de le récompenser pour de bonnes réponses difficiles. C'est pourquoi, en cas de bonne réponse le joueur gagne 10*difficulté_courante points (10 à 100) et en cas de mauvaise réponse il perd 10*(11-difficulté_courante) points (100 à 10).

Une fois que le joueur perd ses quatre vies, la partie se termine et le joueur accède à la troisième et dernière scène du jeu:

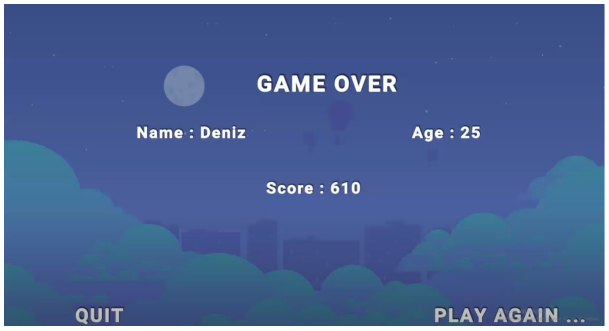

Figure 3: Scène 3

Cet écran de fin permet au joueur de visualiser son score et de relancer une partie en appuyant sur le bouton "Play Again". Par ailleurs, au chargement de cette page la totalité des informations de la partie effectuée sont stockées dans un fichier csv que nous détaillerons dans la section "Résultats".

2.2 Python

Dans cette section, nous détaillerons les technologies utilisées pour capter les signaux et les problèmes rencontrés. L'émotion qui nous intéresse est le stress, nous avons donc jugé pertinent de mesurer le rythme cardiaque de l'utilisateur et son activité électrodermale qui capte la variation électrique de la conductivité de la peau en réponse à la sécrétion de sueur.

Dans un premier temps pour mesurer l'état du joueur nous avons utilisé les outils suivants:

- Grove-GSR-Sensor, mesurant la résistance de la peau.
- Grove-Ear-Clip Heart Rate Sensor, mesurant le battement du joueur dans un intervalle de temps donné.
- Raspberry Pi 3.
- GrovePi Plus.

Le GrovePi est une carte additionnelle compatible avec le raspberry Pi permettant d'utiliser les deux capteurs Grove cités ci-dessus. Avant toutes manipulations, nous avons installé un OS sur le raspberry Pi et tous les frameworks nécessaires à la communication avec le GrovePi. Globalement, nous avons eu beaucoup de mal à trouver une bonne documentation en python pour le GrovePi puisque nos deux capteurs sont généralement utilisés avec le Grove Base Hat ou le Arduino/Seeeduino. En ce qui concerne le Grove-GSR sensor, nous n'avons pas eu de réel problème puisqu'un exemple d'utilisation très similaire était disponible ici. Cependant pour le Grove-Ear-Clip Heart Rate Sensor, nous avons dû adapter l'exemple similaire donné en C++ utilisant la librairie du Grove Base Hat (source) en python puis modifier le code afin d'avoir un temps de mesure beaucoup plus court mais moins précis dans le but d'envoyer le signal au jeu plus fréquemment.

Malheureusement, le premier Grove-Ear-Clip Heart Rate Sensor récupéré s'est avéré défectueux et le deuxième ne fonctionnait plus après seulement quelques utilisations. Par conséquent, nous avons donc décidé de complètement changer le matériel d'acquisition et nous diriger sur les technologies suivantes:

- Bitalino
- The bitalino revolution ecg sensor
- The bitalino electrodermal activity sensor

Cette fois-ci, la documentation du bitalino en python est beaucoup plus fournie et présente plusieurs exemples d'utilisation pour chacun de ses capteurs. De plus, elle propose des outils très intéressants comme open signal qui permet directement de visualiser les signaux, ou même des packages permettant la communication avec le bitalino via Bluetooth sous python.
Une fois nos capteurs branchés et la connexion Bluetooth établie, il suffit de définir le nombre de "sample" à acquérir pour obtenir les valeurs de nos signaux dans un intervalle d'une seconde. Étant donné que le "revolution ecg sensor" renvoie l'ecg, il faut détecter les pics de valeurs pour repérer les battements du coeur. Pour cela, nous utilisons l'algorithme d'Hamilton et calculons le nombre de battements chaque trois secondes et multiplions cette valeur par 20 pour obtenir une approximation rapide du rythme cardiaque. En ce qui concerne la variation électrique de la conductivité de la peau, nous calculons simplement la moyenne des différences des samples obtenus à chaque seconde.

Voici une visualisation de nos deux signaux où l'on distingue les battements du coeur et de légère variation de l'EDA:

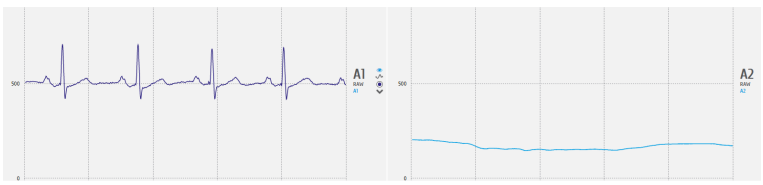

Figure 4: Open signal

Une fois la réception des signaux implémenté, il est nécessaire d'envoyer ces données à Unity pour mettre à jour la difficulté du jeu. Pour cela, nous mettons en place deux threads dans un code python. Le premier calculera une approximation du rythme cardiaque de l'utilisateur durant trois secondes et l'EDA pendant une seconde et mettra à jour des variables globales qui seront envoyées par le deuxième thread. Dans l'objectif d'améliorer la précision du rythme cardiaque sans réduire la vitesse de calcul, nous prenons à chaque itération la moyenne des approximations faite sur les 5 dernières itérations. Dans le cas de l'EDA, c'est uniquement la dernière valeur prise qui nous intéresse puisque celle-ci peut très rapidement être impactée par l'environnement du joueur à un temps précis. Après avoir obtenu nos deux mesures, le processus calculera un score de stress qui permettra d'évaluer si le joueur est plus ou moins stressé. Ce score est donné par:

$$score = HR * (EDA + 0.5)$$

où HR est le rythme cardiaque et EDA la variation électrique de la conductivité de la peau.

Nous avons décidé expérimentalement d'ajouter une constante à l'EDA avant d'effectuer notre multiplication. Puisque celle-ci présentait souvent de très petite valeur, elle prenait très rapidement le dessus sur l'importance du rythme cardiaque dans le calcul de ce score. Or, nous considérons que le rythme cardiaque est un indicateur plus important que l'EDA sur le stress. Cependant, nous somme bien conscients que cette évaluation est très subjective et critiquable, c'est pourquoi nous sommes à l'écoute de toute proposition d'amélioration.

Les signes biologiques étant très différents d'un individu à un autre, il nous était nécessaire d'inclure une phase de préparation de 1 minute avant de lancer le jeu. Durant cette minute, nous calculons un score chaque 4 secondes (3 secondes pour l'ecg et 1 seconde pour l'EDA), à chaque itération nous définissons la valeur maximale et minimale des scores. Avec ceci, il nous est possible de faire un mapping de ce score sur l'intervalle [0,0.3]:

$$MappedScore = (0.3/(maxScore - minScore) * (score\_courant - minScore))$$

La borne maximale de 0.3 a été choisis de manière pragmatique en prenant une valeur pas trop grande qui ne changerait pas l'état du jeu de manière trop brusque mais qui tout de même se fera ressentir au bout d'un certain nombre d'équations. Cependant, ce n'est pas exactement ce nombre qui est envoyé à Unity. Puisque nous voulons que le jeu devienne moins rapidement difficile lorsque l'utilisateur est stressé, il est nécessaire que le score envoyé soit petit lorsque le joueur est stressé. Pour ce faire il suffit simplement de soustraire à la borne maximale le score courant:

$$SendScore = 0.3 - MappedScore$$

Étant un serveur constamment à l'écoute de requête faite par Unity, le deuxième thread envoie une liste contenant le dernier SendScore, HeartRate et EDA calculé à chaque changement d'image de fond sur Unity. C'est à ce moment que la difficulté du jeu est mis à jour:

$$Difficulte = Difficulte\_courante + Constante + SendScore$$

Encore une fois la constante a été définie à 0.2 de façon expérimentale de sorte à ce que le jeu devienne tout de même de plus en plus dur même si l'utilisateur se montre extrêmement serein durant toute la partie.
Lorsque la difficulté atteint le seuil de 7.5 (comme mentionné dans la partie Unity, c'est à ce moment-là que les équations possèdent trois termes), la vitesse de défilement redescend à une valeur de 2.0 (max = 3.0) et ne dépend plus complètement de la difficulté mais uniquement de ce score envoyé:

$$MouvementSpeed = MouvementSpeed + SendScore$$

Avec la borne maximale de la vitesse qui reste à 3.0.

L'explication du fonctionnement de tout un système étant difficile à l'écrit voici maintenant un visuel sur l'architecture complète:

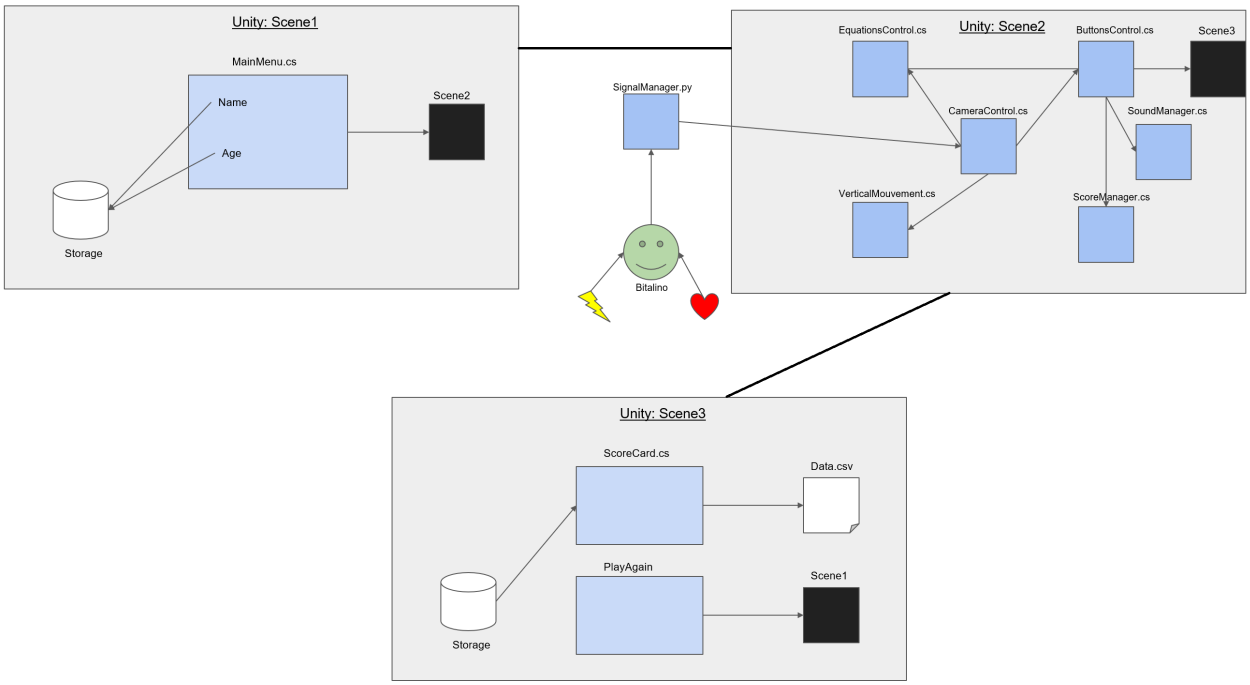

Figure 5: Architecture

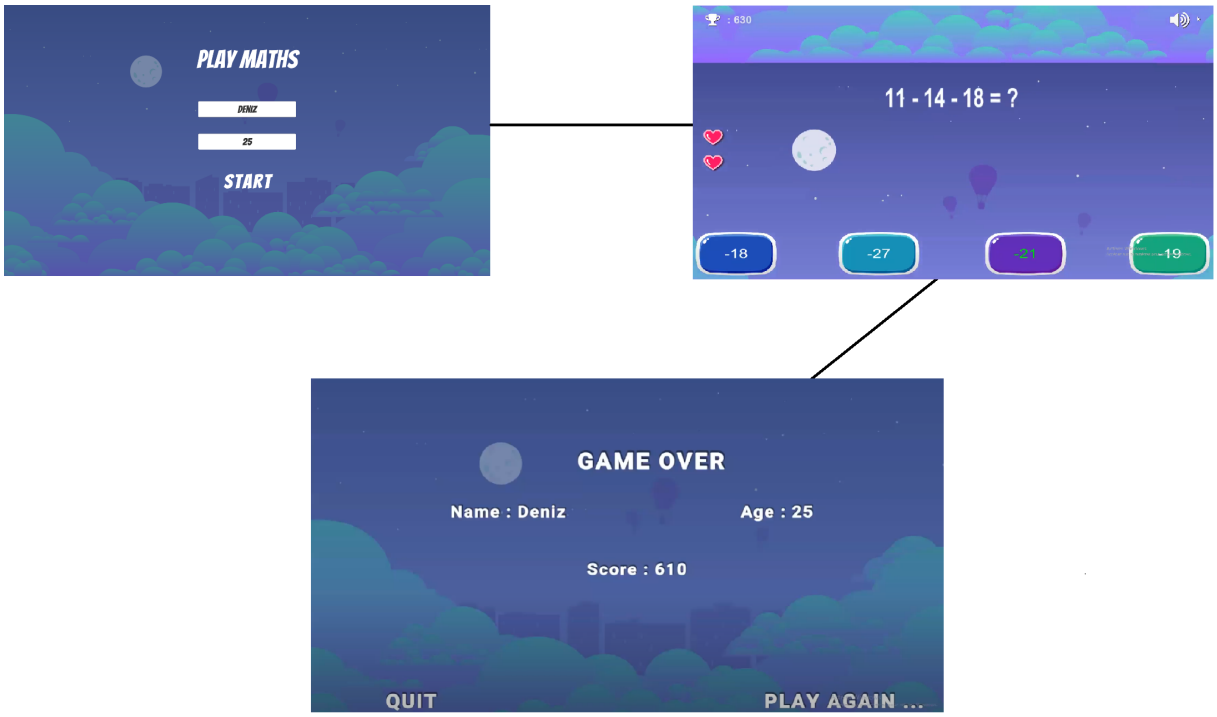

Figure 6: Différentes scènes

3 Résultats

À la fin d'une partie, l'ensemble des informations est stocké dans un fichier csv où à chaque ligne on retrouve:

- Le timestamp indiquant le moment où la requête d'unity a été faite.
- Le score envoyé par le code python.
- Le rythme cardiaque
- L'EDA
- Le score

Pour chaque utilisateur, on vérifie s'il existe un csv associé. Si oui, alors on ajoute un délimiteur qui indique le commencement d'une nouvelle partie, sinon on crée un nouveau csv dont le nom indique l'âge de l'utilisateur et son prénom qui est précédé d'un "b" si celui-ci utilise le Makey-Makey. Ensuite, pour chaque équation proposée une ligne s'ajoute dans le csv avec les informations nécessaires:

1	timestamp	mappedScore	HR	EDA	Score
2	11.06.2021 23:33:12	0.08053329	80	0.1581582	10
3	11.06.2021 23:33:22	0.05382627	85	0.1461461	20
4	11.06.2021 23:33:31	0.03007903	86.66666	0.1651652	30
5	11.06.2021 23:33:41	0.1469699	80	0.1051051	40
6	11.06.2021 23:33:49	0.2354991	66.66666	0.1011011	60
7	11.06.2021 23:33:57	0.1352481	75	0.1601602	80
8	11.06.2021 23:34:05	0.1384643	80	0.1151151	100
9	11.06.2021 23:34:12	0.02948612	86.66666	0.1861862	130
10	11.06.2021 23:34:19	0.1086946	80	0.1501502	160
11	11.06.2021 23:34:26	0.1342115	80	0.1201201	190

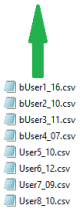

Figure 7: Fichiers csv

Après avoir fait essayer notre jeu à 8 jeunes individus âgés de 7 à 16 ans, nous avons obtenu pour chaque utilisateur un fichier csv possédant les informations des quatre parties jouées. Puis, nous avons donc implémenté un code python qui lit tous ces fichiers et dessine les graphes suivants:

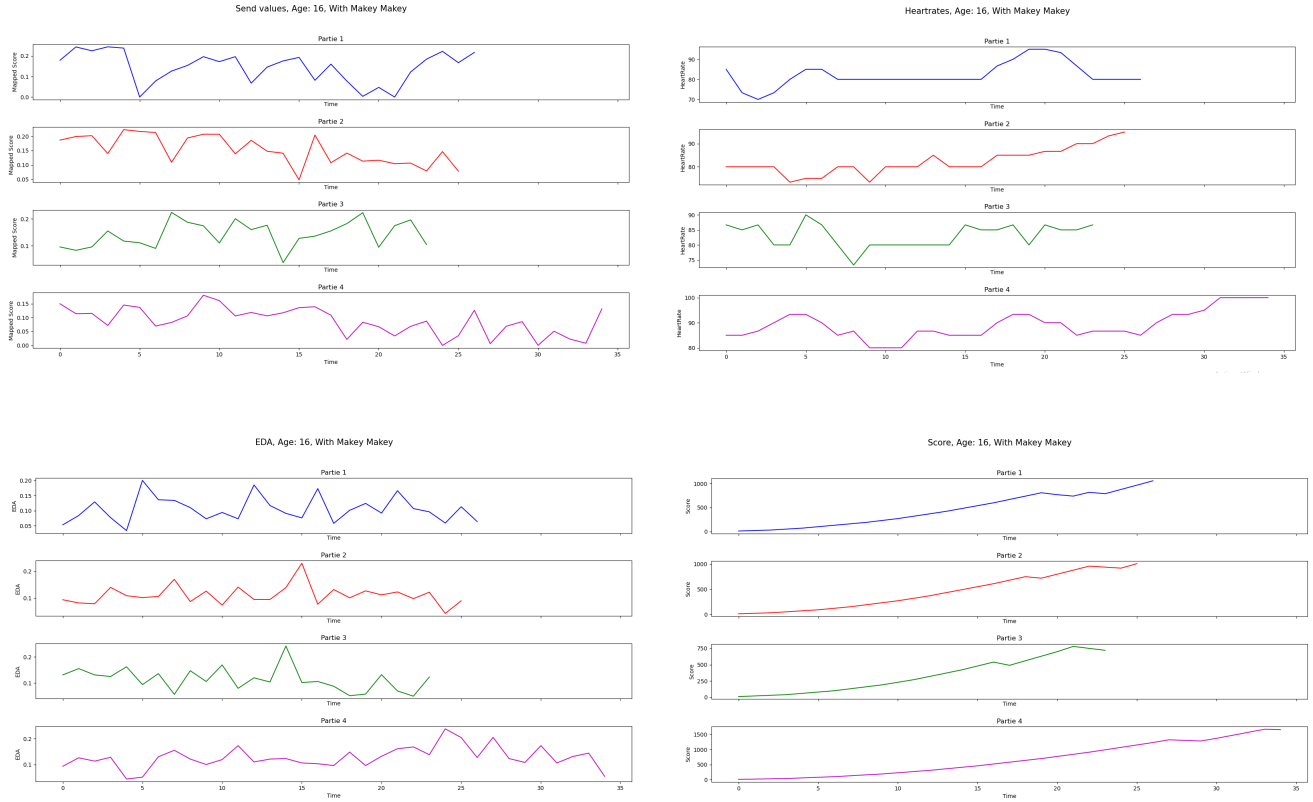

Figure 8: Utilisateur 1

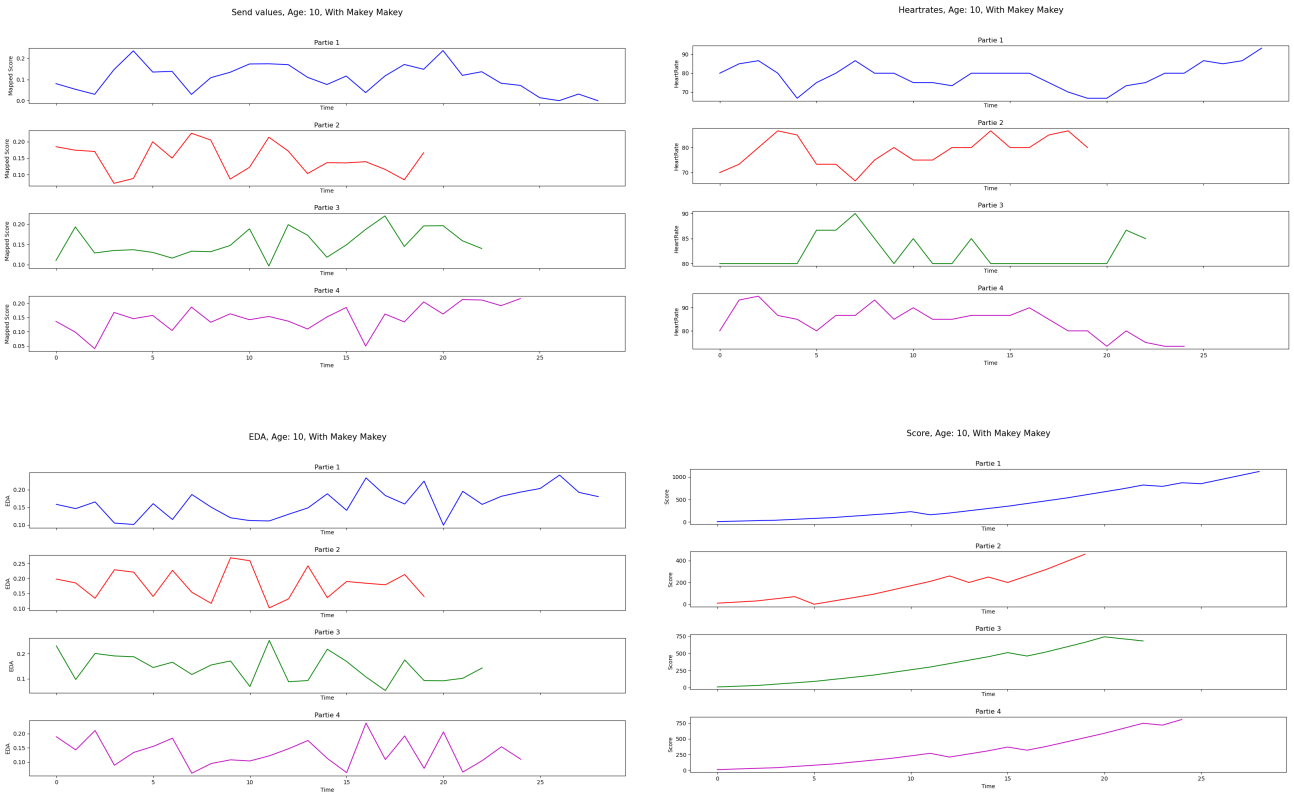

Figure 9: Utilisateur 2

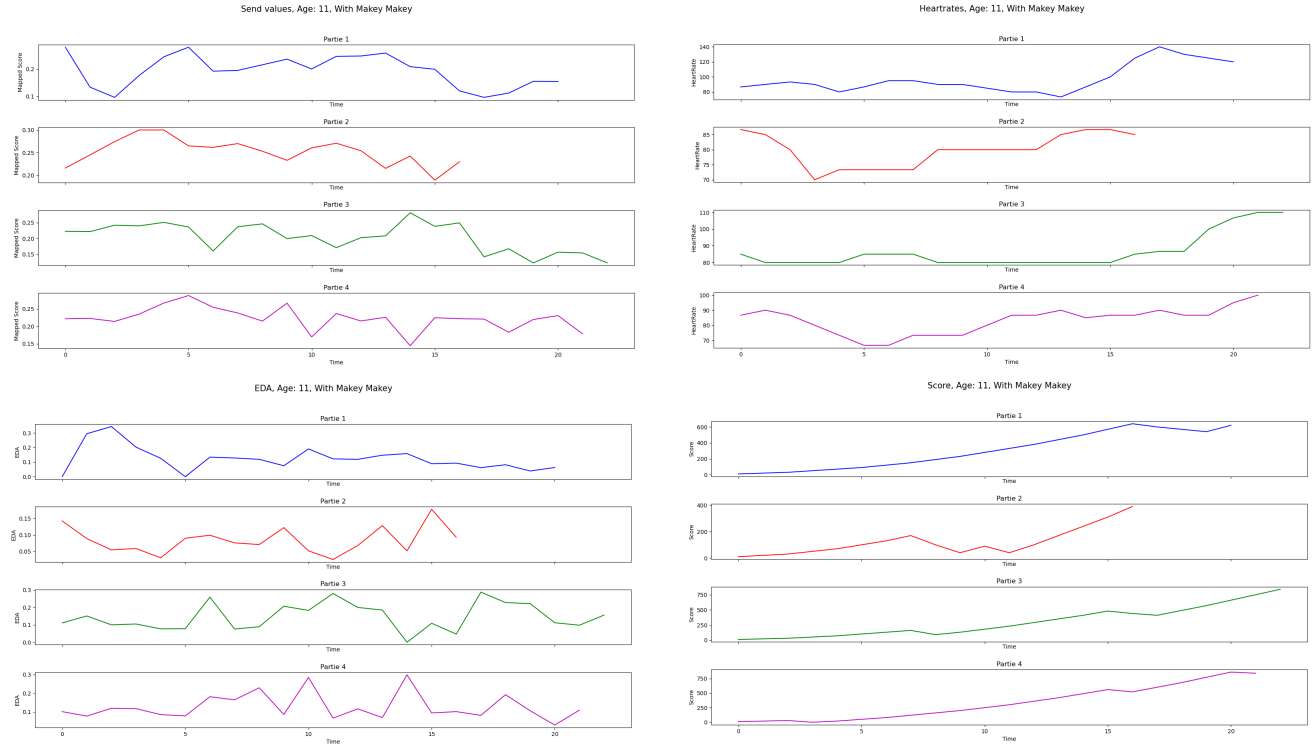

Figure 10: Utilisateur 3

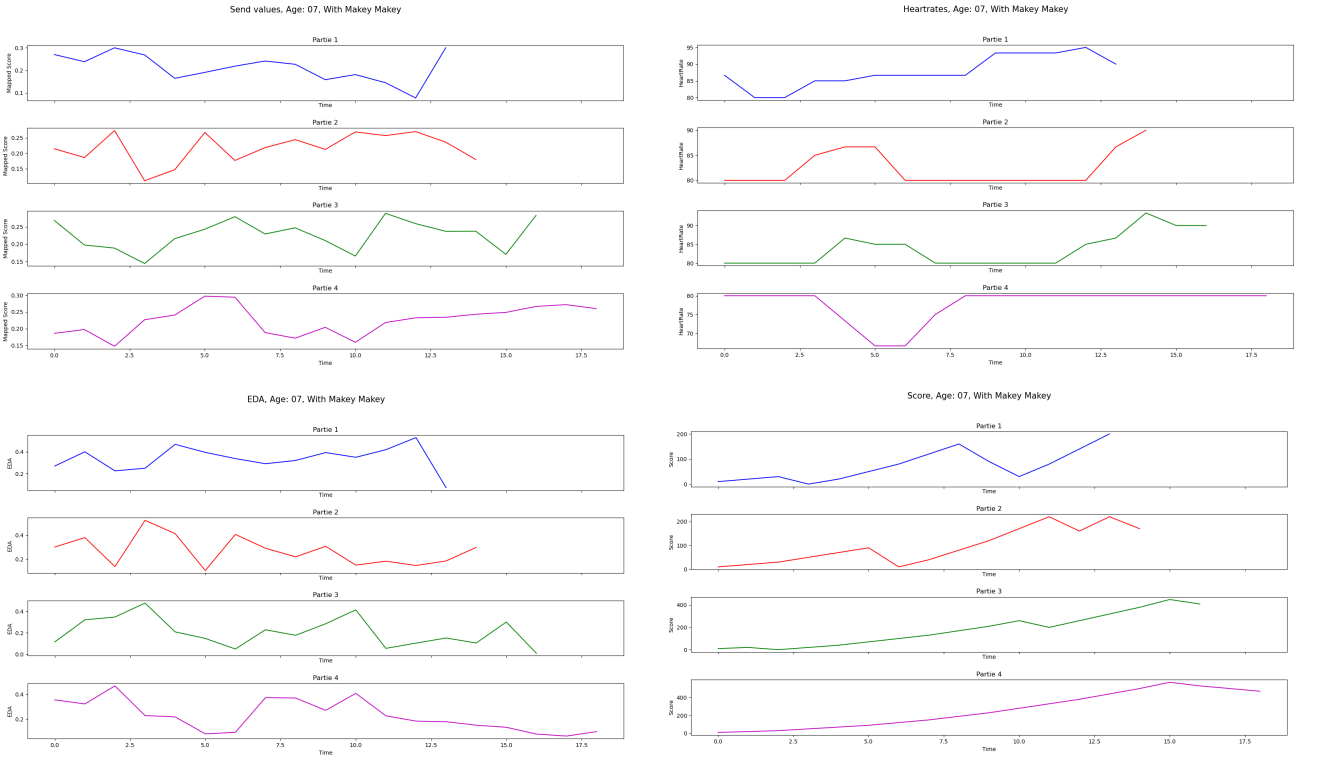

Figure 11: Utilisateur 4

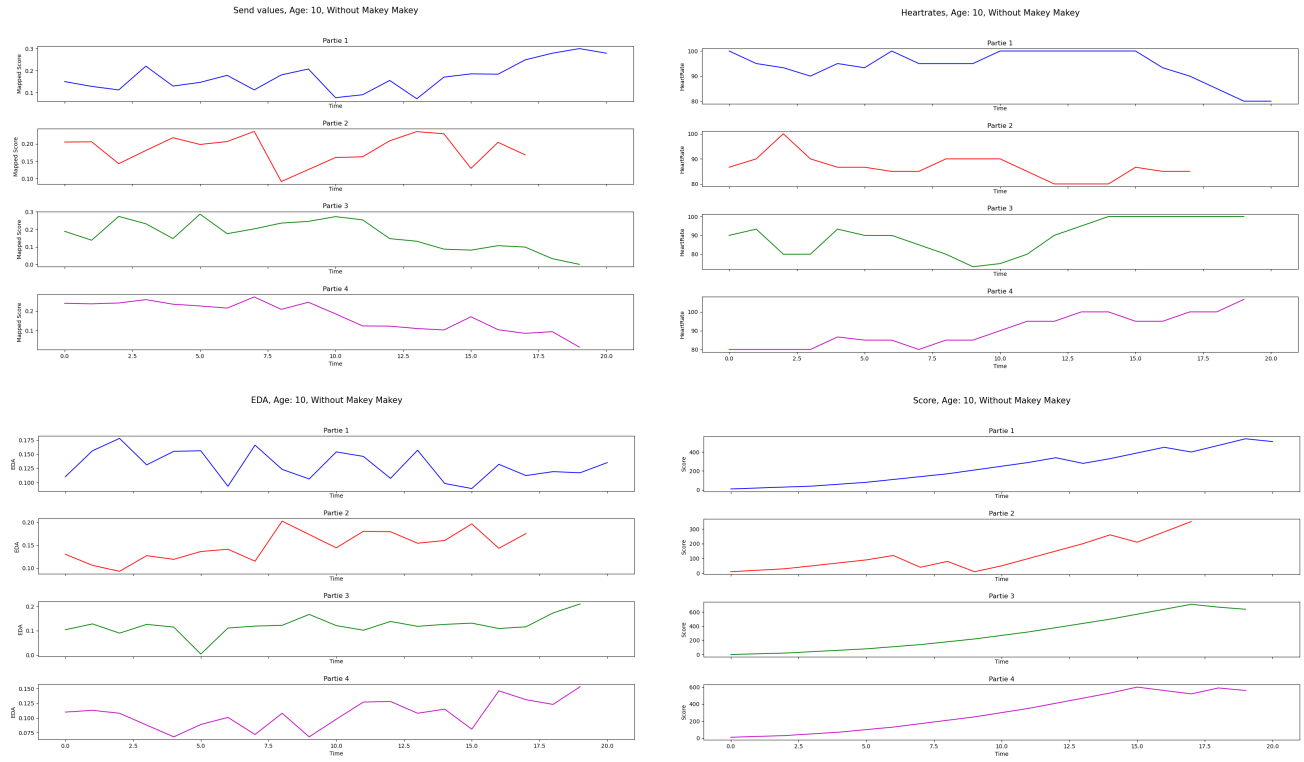

Figure 12: Utilisateur 5

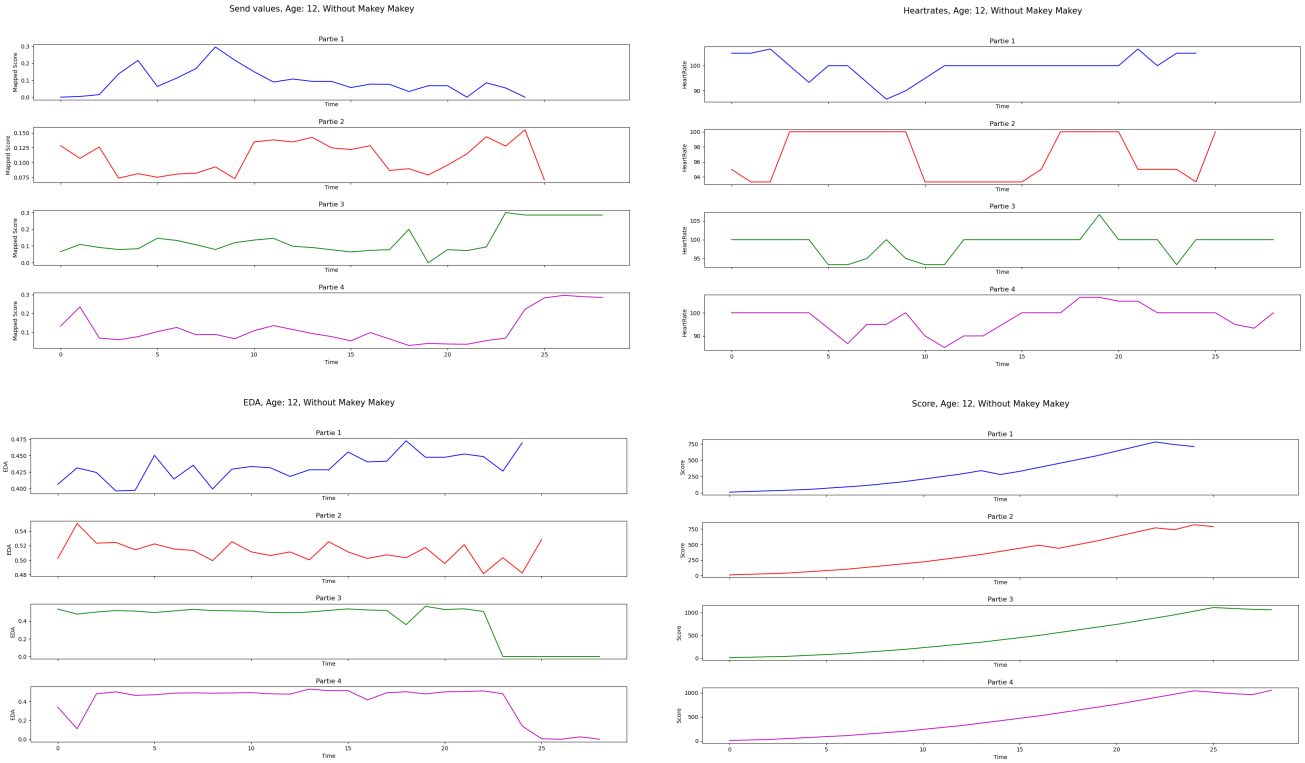

Figure 13: Utilisateur 6

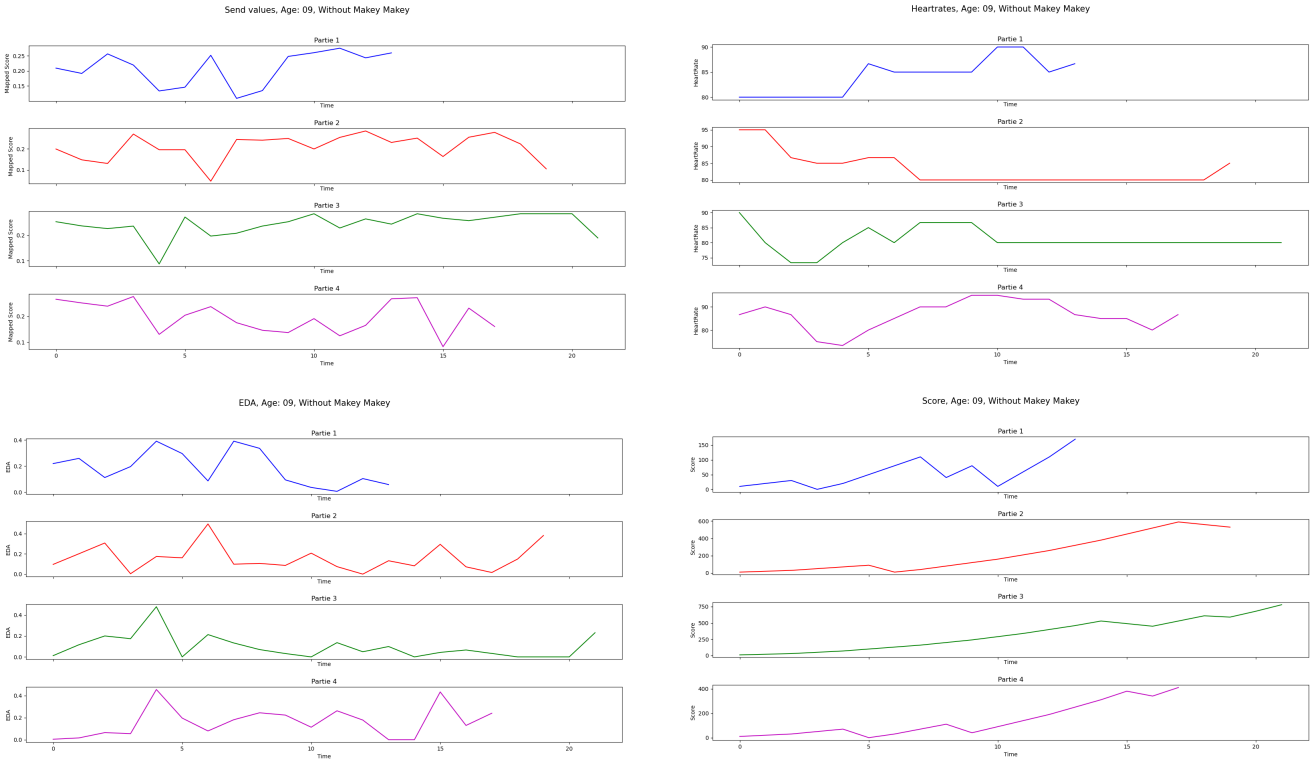

Figure 14: Utilisateur 7

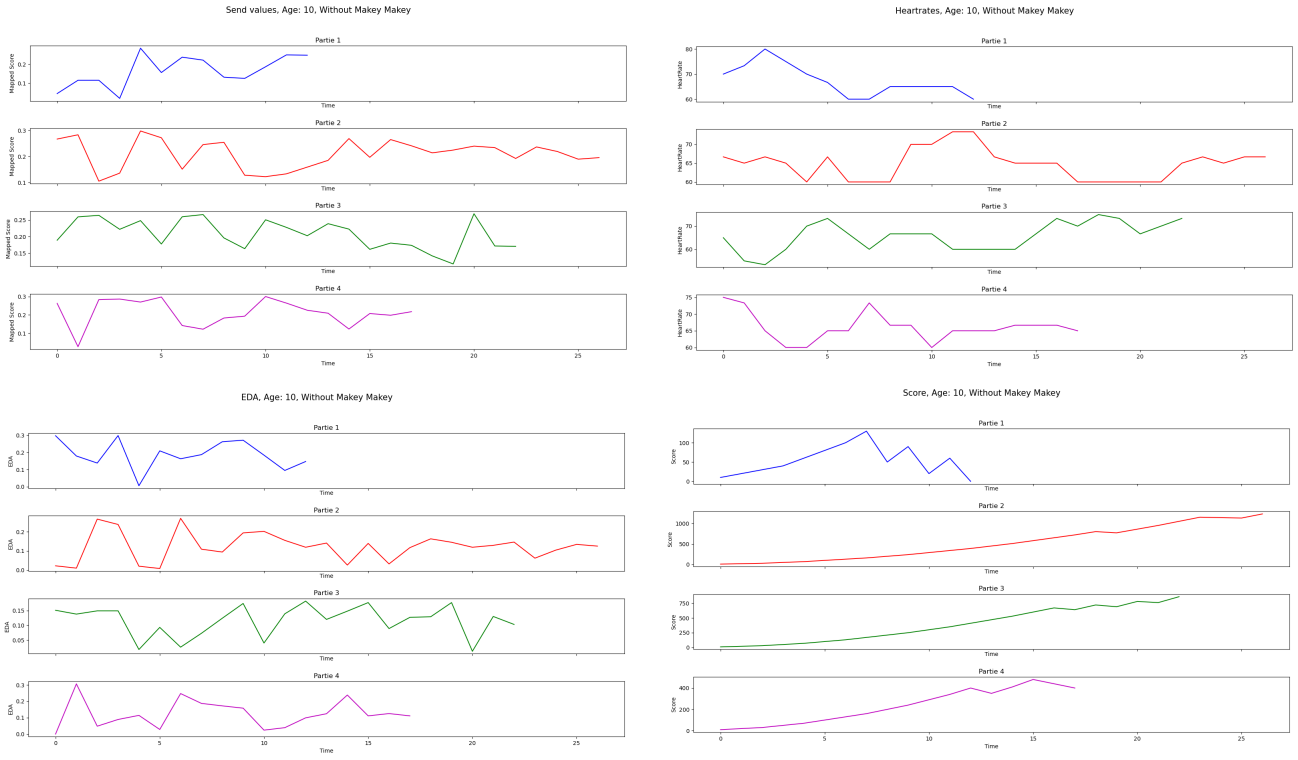

Figure 15: Utilisateur 8

Globalement, on observe dans la plupart des cas une augmentation du rythme cardiaque au fil de la partie. Cependant, le comportement de l'EDA paraît très aléatoire, il est très difficile de trouver un pattern commun puisque ce capteur est très sensible à l'environnement du joueur qui n'est pas réellement pris en compte ici.
Puisqu'il est très difficile de trouver 8 participants du même âge en deux semaines, nous avons décidé de prendre des candidats âgés de 7 à 16 ans. Généralement, la capacité intellectuelle d'individu de jeunes âges varie énormément d'une année à une autre, c'est pourquoi nous nous attendions à une forte disparité dans les scores obtenus. Comme l'indiquent les graphiques, nos observations montrent un meilleur score moyen chez les individus les plus âgés.

En ce qui concerne l'amélioration du score des joueurs à travers les quatre parties, 5 personnes sur 8 montrent une progression linéaire. Ce constat reste positif mais il est difficile d'en déduire un réel signe d'efficacité du système sur un si petit échantillon. De plus, sur ces 5 personnes trois candidats ont utilisé le Makey-Makey ce qui n'est pas assez significatif pour déduire la moindre différence.
D'autre part, les meilleurs scores obtenus montrent souvent un rythme cardiaque médian qui ne varie que très peu jusqu'à la fin de la partie où le rythme cardiaque devient souvent maximal. Cela soutient bien notre hypothèse supposant qu'il ait tout de même nécessaire d'avoir un certain niveau de stress optimal pour la réalisation d'une tâche. Si ce niveau de stress se montre trop bas cela peut signifier un relâchement total de l'utilisateur sur la résolution des équations. Au contraire, si ce niveau devient trop haut c'est que l'utilisateur perd ses moyens au vue de la difficulté trop grande des équations affichées.

Nous avons tout de même essayé d'avoir une moyenne d'âge similaire pour nos deux groupes, afin de tenter une comparaison des scores obtenus. Celui avec le Makey-Makey a une moyenne d'âge de 11 ans contre 10.25 sans le Makey-Makey. Le premier groupe montre une moyenne de score de 716 et une moyenne de meilleur score de 1035 contre 628 et 927,5 pour le deuxième groupe. À priori, il n'y a donc pas vraiment de réel avantage à utiliser le Makey-Makey, car cette légère différence peut s'expliquer par la différence d'âge entre les deux groupes ou même une différence de niveau entre des individus de même âge.

4 Conclusion

En résumé, la quasi-totalité de nos hypothèses s'est confirmée pour le petit nombre de personnes testé. Toutefois, le Makey-Makey n'a pas montré l'impact escompté. Nous pensions que cela favoriserait un meilleur score en motivant les jeunes à plus s'investir dans le jeu et en réduisant le temps de réaction pour choisir une réponse.
De manière plus générale, ce projet nous a beaucoup apporté en terme de travail collectif. Pour pouvoir développer cette application, nous avons d'abord dû visualiser sur la forme comme sur le fond notre application. Cela nous a permis de diviser le travail en un ensemble de sous-tâche planifié que nous avons essayé de réaliser dans le temps prévu.
L'élaboration de cette application nous a permis de bonifier notre capacité à travailler de manière autonome, car nous avons dû chercher et comprendre les bonnes documentations afin d'implémenter les fonctionnalités désirées. De plus, ce travail ayant été réalisé à deux, il peut arriver que des avis divergent sur certaines manières d'aborder un problème, mais nous avons su partager le travail de manière équitable et intelligente pour maximiser au mieux notre productivité. La réalisation de ce projet nous a également contraints à fournir un effort régulier pour parvenir à un résultat satisfaisant utilisant de nouvelles technologies intéressantes comme Unity ou le bitalino.
Pour conclure, voici le lien de notre Github contenant l'intégralité de notre projet et d'une démonstration vidéo du jeu.