Objectifs

- Affichage de l'arrière plan
- Lecture des touches du clavier
- Déplacement du joueur en fonction de l'état du clavier

Il est impératif de valider chaque étape avant de passer à la suivante.

En cas de doute ou de difficulté, consultez votre enseignant.

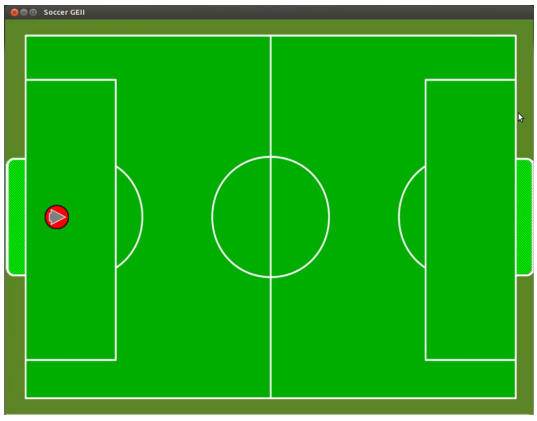
Affichage de l'arrière plan

Dans le répertoire images, un fichier playground.png vous est fourni. Créez un sprite nommé PlayGround associé à cette image. Dans la partie du programme principal dédiée à la gestion des graphismes, insérez le positionnement et l'affichage de l'arrière plan. Pensez à utiliser les constantes symboliques précédemment déclarées. Vous veillerez à afficher l'arrière plan avant les autres éléments afin que ceux-ci restent visibles.

Avant de poursuivre, replacer votre joueur à sa position initiale au début du programme :

```
PlayerProperties PropPlayer1={ 100. , WINDOW_HEIGHT/2. , 0. , 0.};
```

Les éléments graphiques doivent être disposés exactement comme dans l'illustration ci-dessous. Le joueur 1 doit être immobile.



Gestion des événements

Nous allons nous intéresser à la partie du programme permettant la gestion des événements. Un événement est comparable à un front montant ou descendant: quelque chose vient de changer d'état. Cela peut-être un appui sur une touche, un mouvement de la souris, la fermeture de la fenêtre, etc …

Voici un extrait du programme principal:

```
// Effectue le traitement de chaque événement
sf::Event Event;
while (App.GetEvent(Event))
{
        // Si la fenêtre est fermée, on ferme l'application
        if (Event.Type == sf::Event::Closed) App.Close();
}
```

Dans cet extrait, une variable événement est déclarée, ensuite tant qu'il y a des événements en attente, ceux-ci sont traités. C'est le rôle de la boucle while. La fonction App.GetEvent(Event) renvoie vrai s'il y a au moins un événement en attente. Dans le corps de la boucle, il faut vérifier le type d'événement et agir si nécessaire. Dans le cas présent, un seul événement est testé: la fermeture de la fenêtre (Event.Type == sf::Event::Closed), qui engendre la fermeture de l'application (App.Close()). Supprimer cette ligne, lancer l'application et essayer de fermer la fenêtre.

Le gestionnaire d'événements pourrait être utilisé pour lire les touches du clavier, mais il est mal approprié pour une lecture rapide des touches. Pour cela, nous n'allons pas attendre l'événement, mais tester directement l'état de la touche : enfoncée ou non. La fonction App.GetInput().IsKeyDown(ToucheATester) renvoie *vrai* si la touche à tester est enfoncée.

Ajouter le test ci-dessous après la boucle de gestion des événements et tester son fonctionnement.

```
// Un appui sur Echap permet de quitter l'application
if (App.GetInput().IsKeyDown(sf::Key::Escape)) App.Close();
```

Dans Qt-creator, cliquez sur mot clef *Escape* de la ligne ci-dessus en maintenant la touche Ctrl enfoncée. Vous avez la liste des touches utilisables. Cherchez et repérez les touche de directions (flèches haut, bas, gauche et droite).

Gestion de la rotation

Après avoir testé un appui sur la touche Echap, ajoutez le test des flèches de direction droite et gauche avec la règle suivante :

- un appui sur la flèche *gauche* augmente l'angle de rotation de ROTATION_STEP,
- un appui sur la flèche *droite* diminue l'angle de rotation de ROTATION_STEP.

ROTATION_STEP est une constante symbolique définie dans le fichier tools.h.

```
// Paramètres des déplacements
#define         ROTATION_STEP           0.05
```

Tester votre ajout en vérifiant que le joueur 1 tourne lorsque l'on appuie sur les flèches (et uniquement dans ce cas).

Gestion de la vitesse

Sur le même principe, ajoutez le test des flèches de direction haut et bas avec la règle suivante :

- un appui sur la flèche *haut* augmente la vitesse de SPEED_STEP,
- un appui sur la flèche *bas* diminue la vitesse de SPEED_STEP.

ROTATION_STEP est une constante symbolique définie dans le fichier tools.h et qui vaut 0,1. Tester votre modification en vérifiant que le joueur 1 accélère et ralentit correctement. Vous constaterez que le joueur peut avancer mais aussi reculer, ce qui n'est pas souhaitable ici. Dans la fonction ComputeDynamic, vous allez brider la vitesse entre 0 et SPEED_MAX avant de calculer le modèle:

- si la vitesse est négative, celle-ci est ramenée à 0,
- si la vitesse est supérieur à SPEED_MAX, celle-ci est ramenée à SPEED_MAX.

SPEED_MAX sera défini à 5. Vérifier que vous ne pouvez plus reculer.

Limites du terrain

Il ne vous aura pas échappé non plus que le joueur peut sortir de la fenêtre. Pour résoudre ce problème, commencez par créer, dans tools.h, 4 constantes symboliques :

- LIMIT_TOP
- LIMIT_BOTTOM
- LIMIT_LEFT
- LIMIT_RIGHT

Dans la fonction ComputeDynamic, après le calcul du modèle, vérifier que le joueur n'est pas sorti de la fenêtre en vous inspirant de la vérification de la bordure gauche fournie ci-dessous :

```
if (prop->x<LIMIT_LEFT)     { prop->x=LIMIT_LEFT; prop->speed=0; }
```

Vous noterez qu'une sortie du terrain a deux conséquences : le replacement du joueur à l'intérieur, mais également l'arrêt du déplacement (remise à zéro de la vitesse).

Comme le terrain est plus petit que la fenêtre, vous allez devoir chercher de façon empirique les valeurs numériques des limites. Pour éviter quelques désagréments, notez que le compilateur ne compile que les librairies dont le .cpp a été modifié depuis la dernière compilation. Si vous changer uniquement une constante dans un fichier .h, la modification ne sera pas prise en compte lors de la compilation. Pour pallier à ce problème, choisissez dans le menu *Compiler* la commande *Tout recompiler.*

Commentez, indentez et faites valider par l'enseignant.