READ ME

of FRC2018 simulator game play

Introduction

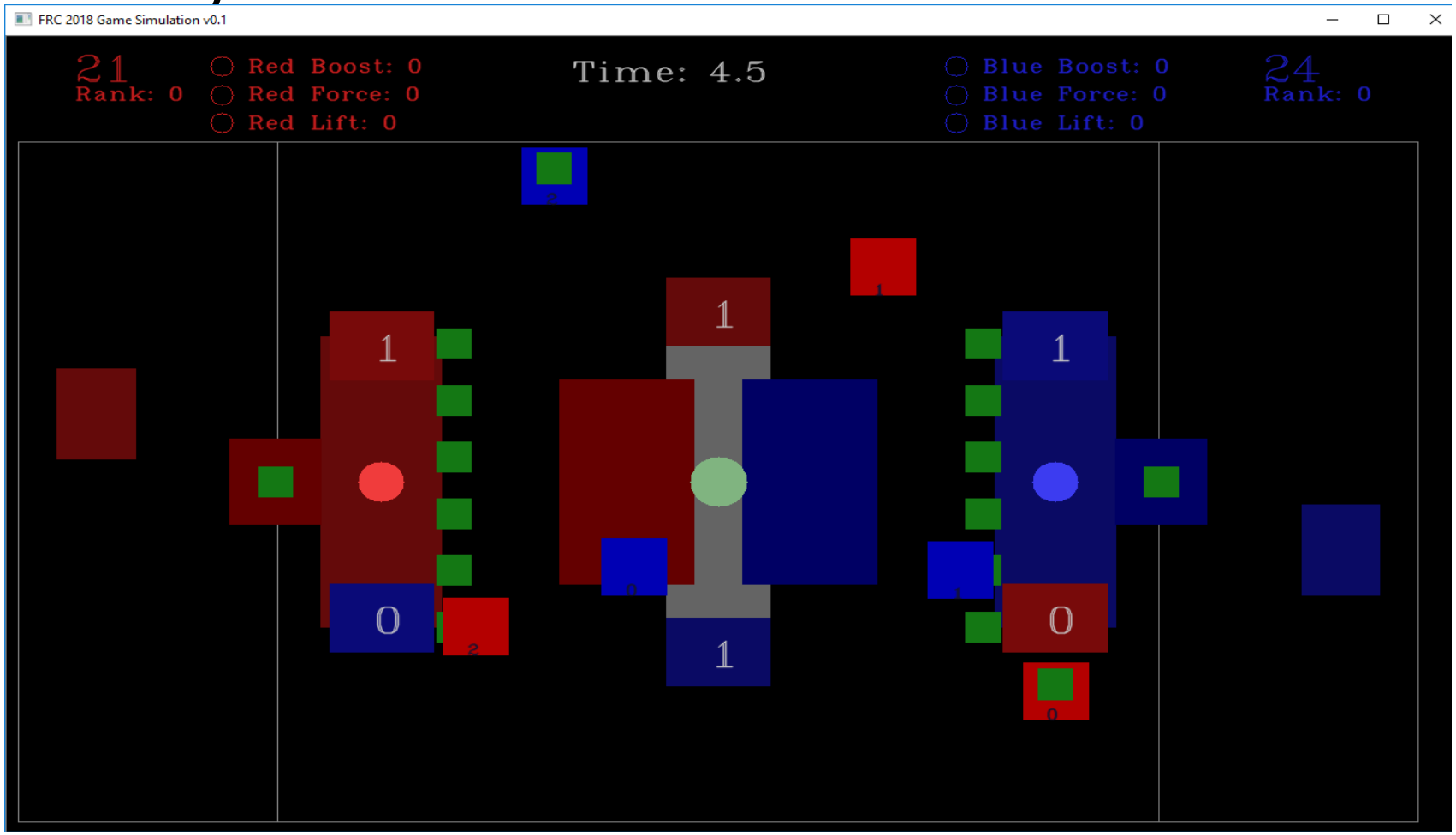
- It is a computer game designed to mimic FRC2018 robot competition
- In this game, player programs three blue robots to compete three AI controlled red robots.
  - Player must put her strategy into C++ functions, then
  - Sit back and relax to watch her functions play the game
- Following pages show how to program robots in this game
- But, before continue on following pages, please try to run my pre-programmed example
  - simulator.exe is a pre-build executable for Windows
  - simulator is a pre-build executable for Ubuntu with openCV3.3 installed.

Build Environment

- Ubuntu 16.04 with OpenCv 3.3
  - Both Makefile and Eclipse project files are available
- Windows with visual studio 2017 and OpenCv 3.3
  - Visual Studio solution file

- Please let me know if you want to build it on other environment.

Game Play -- watch and relax



Sorry, it is just a still image, not video, because I don't want upload a big file.

Program Your Robots

- Three robots are programed by three header files,
  - robot_blue0.h, robot_blue1.h and robot_blue2.h
  - A player is expected to write one function, getNextAction(), for each robot

Example -- robot_blue1.h

- In my example, blue robot 1 is assigned to take care blue side switch and use ramp to lift other robots.
- Because robot 1 cannot lift itself on the platform, it must use Lift Vault.
- I also make sure that my assigned tasks are always feasible.

Function Prototype

```
virtual
void
getNextAction
(
        platform *pPlatformInOut,
        searchActionType * pActionOut
)
```

virtual function to replace the base class implementation

pPlatformInOut has the current competition state, for example, the number of blocks on scale, game time and position of each robot.

The output of your program. It is the next action of your robot.

Get the Current Game Play State

```
double currentTime = pPlatformInOut->getTime();
```

Get the current time

```
const platformStateType *pPlatformState = pPlatformInOut->getState();
```

platformStateType has cube count of each switch and scale.

```
coordinateType robotPosition = pPlatformInOut->getRobotPos(ALLIANCE_BLUE, m_robotIndex);
```

The position of the current robot

```
coordinateType rampRobotDestination = pPlatformInOut-> getBlueLiftZonePosition();
```

The position to lift other robots

Note: Please look up platform.h for any information of the current game play state.

Make Decision Based on Time and State (1)

```
initTaskToNoAction(pActionOut);
```

Initialize the output data structure.
The initialized default action is NO ACTION.

```
if (currentTime > COMPETITION_END_TIME) {
    if ((pRobotState->pos.center.y == rampRobotDestination.y) &&
        (pRobotState->pos.center.x == rampRobotDestination.x)) {
          pActionOut->actionType = INVALID_ACTION;
          return;
    }
} else
```

COMPETITION_END_TIME means start of lifting

If the robot is at lifting ramp position, it cannot do anything,
just return the default action.
Note: INVALID_ACTION means no action.

Make Decision Based on Time and State (2)

It is not at the ramp position

```
else {
    pActionOut->actionType = BLUE_ROBOT_GOTO_POS;
    pActionOut->actionDonePos = rampRobotDestination;

    if (checkIfActionFeasible(ALLIANCE_BLUE,      //alliance name
                              pPlatformInOut,     //platform object
                              pActionOut)) {      //output action plan
        m_idleCount = 0;
        return;
    }
}
```

Issue go to position command Please see config.h for all possible robot actions.

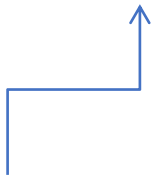
Check if the action is feasible, for example, is there any other robots blocking the path way.

If it is feasible, reset robot idle counter and return with GOTO action.

Note: m_idleCount is the number of contiguous idle cycles.

Second Priority Action

```
pPlannedAction = getPlannedAction();
if (pPlannedAction->actionType != INVALID_ACTION) {
    return;
}
```



If lifting is not feasible, or it is not time for lifting, continue whatever action the robot is doing, unless the robot is idle.

Third Priority Action

If too many cubes on the switch, we may want run a lower priority action.

```
if (pPlatformState->switchBlue_BlueBlockCount <
    pPlatformState->switchBlue_RedBlockCount + 2 + m_idleCount/4) {

      pActionOut->actionType = CUBE_BLUE_OFFENSE_SWITCH;
      //check if the action is feasible
      if (checkIfActionFeasible(
                  ALLIANCE_BLUE,      //alliance name
                  pPlatformInOut,     //platform object
                  pActionOut)) {      //output action plan

                  m_idleCount = 0;
                  return;
      }
}
```

But, if the robot is idle too many cycles, it is no harm to add more cubes

Put a cube on the blue switch

Fourth Priority Action

```
if (pPlatformState->liftBlueBlockCount < 3) {

    pActionOut->actionType = CUBE_BLUE_LIFT_VAULT;

    if (checkIfActionFeasible(
        ALLIANCE_BLUE,      //alliance name
        pPlatformInOut,     //platform object
        pActionOut)) {      //output action plan

        m_idleCount = 0;
        return;
    }
}
```

Add a cube to the lift vault

Note: It is a very simple simulator. If there are 3 cubes in the vault, the lift button will automatically push.

But, it is not true with boost and force buttons. You must program the timing of button push yourself.

Go To Idle

```
m_idleCount++;
```

If no action is feasible, go to idle and increase the idle counter.

```
//stay close to offense switch for quick response
pActionOut->actionType = BLUE_ROBOT_GOTO_POS;
pActionOut->actionDonePos = { 600, 250 };
```

Go to a position for quick response on the next cycle.

Note: If the GOTO command is not feasible, the system will automatically reject this command and the robot will receive NO ACTION command instead.

Config Robot Speed

- Please use BLUE_CONFIGURATION in config.h to config the speed of each robot.
  - I am using faster speed with blue robot to beat red robots. But, don’t follow me! Please use strategy, not speed to win.

All Comments and Suggestions are Welcome