```cpp
#include "examplebot.h"

#include <ctime>
#include <math.h>
#include <string>
#include <assert.h>
#include <iostream>

#include "rlbot/bot.h"
#include "rlbot/color.h"
#include "rlbot/interface.h"
#include "rlbot/rlbot_generated.h"
#include "rlbot/scopedrenderer.h"
#include "rlbot/statesetting.h"

#define PI 3.1415

ExampleBot::ExampleBot(int _index, int _team, std::string _name)
    : Bot(_index, _team, _name) {
}

ExampleBot::~ExampleBot() {
    // Free your allocated memory here.
}

rlbot::Controller ExampleBot::GetOutput(rlbot::GameTickPacket gametickpacket) {

    if (!initialized) {
        initialized = InitializeFalken();
    }

    bool training_complete =
        //(gametickpacket->gameInfo()->secondsElapsed() > snapTime);
        session->training_state() == falken::Session::kTrainingStateComplete;
    switch (session->type()) {
    case falken::Session::kTypeInteractiveTraining:
        if (training_complete) {
            //snapTime += 60;
            StartEvaluationSession();
        }
            break;
    case falken::Session::kTypeEvaluation:
        if (training_complete) {
            //snapTime += 60;
            StartInferenceSession();
        }
        break;
    default:
        break;
    }

    auto& brain_spec = brain->brain_spec_base();
```

```
53
54      rlbot::flat::Vector3 ballLocation =
55          *gametickpacket->ball()->physics()->location();
56      rlbot::flat::Rotator ballRotation =
57          *gametickpacket->ball()->physics()->rotation();
58      rlbot::flat::Vector3 carLocation =
59          *gametickpacket->players()->Get(index)->physics()->location();
60      rlbot::flat::Rotator carRotation =
61          *gametickpacket->players()->Get(index)->physics()->rotation();
62
63      //falken::Rotation falk_car_rot = falken::Rotation::FromEulerAngles(
64      //  carRotation.pitch(), carRotation.yaw(), carRotation.roll());
65      //falken::Rotation falk_ball_rot = falken::Rotation::FromEulerAngles(
66      //  ballRotation.pitch(), ballRotation.yaw(), ballRotation.roll());
67
68      //vec3 car_r = vec3({ carRotation.pitch(),carRotation.yaw(),carRotation.roll
          () });
69      vec3 car_r = vec3({ 0,carRotation.yaw(),0 });
70      mat3 car_m = euler_to_rotation(car_r);
71      quaternion car_quat = rotation_to_quaternion(car_m);
72
73      vec3 ball_r = vec3({ ballRotation.pitch(),ballRotation.yaw
          (),ballRotation.roll() });
74      //vec3 ball_r = vec3({ 0,0,0 });
75      mat3 ball_m = euler_to_rotation(ball_r);
76      quaternion ball_quat = rotation_to_quaternion(ball_m);
77
78      falken::Position falk_car_pos = falken::Position({
79          carLocation.x(),carLocation.z(),-1 * carLocation.y() });
80      falken::Rotation falk_car_rot = falken::Rotation({
81          car_quat[0],car_quat[1],car_quat[2],car_quat[3] });
82      falken::Position falk_ball_pos = falken::Position({
83          ballLocation.x(),ballLocation.z(),-1 * ballLocation.y() });
84      falken::Rotation falk_ball_rot = falken::Rotation({
85          ball_quat[0],ball_quat[1],ball_quat[2],ball_quat[3] });
86
87
88      brain_spec.observations_base().position.set_value(falk_car_pos);
89      brain_spec.observations_base().rotation.set_value(falk_car_rot);
90      brain_spec.observations_base().entity("entity_0")->position.set_value
          (falk_ball_pos
91      );
92      brain_spec.observations_base().entity("entity_0")->rotation.set_value
          (falk_ball_rot);
93
94      // Calculate the velocity of the ball.
95      //float velocity = sqrt(ballVelocity.x() * ballVelocity.x() +
96      //                      ballVelocity.y() * ballVelocity.y() +
97      //                      ballVelocity.z() * ballVelocity.z());
98
99      // This renderer will build and send the packet once it goes out of scope.
100     rlbot::ScopedRenderer renderer("test");
```

```
101
102     // Load the ballprediction into a vector to use for rendering.
103     //std::vector<const rlbot::flat::Vector3 *> points;
104
105     //rlbot::BallPrediction ballprediction = GetBallPrediction();
106
107     //for (uint32_t i = 0; i < ballprediction->slices()->size(); i++) {
108     //  points.push_back(ballprediction->slices()->Get(i)->physics()->location
        ());
109     //}
110
111     //renderer.DrawPolyLine3D(rlbot::Color::red, points);
112
113     rlbot::Controller controller{ 0 };
114
115     if (!brain_control) {
116         renderer.DrawString2D("ATBA CONTROL", rlbot::Color::green,
117             rlbot::flat::Vector3{ 10, 10, 0 }, 4, 4);
118         // Calculate to get the angle from the front of the bot's car to the
          ball.
119         double botToTargetAngle = atan2(ballLocation.y() - carLocation.y(),
120             ballLocation.x() - carLocation.x());
121         double botFrontToTargetAngle = botToTargetAngle - carRotation.yaw();
122         // Correct the angle.
123         if (botFrontToTargetAngle < -PI)
124             botFrontToTargetAngle += 2 * PI;
125         if (botFrontToTargetAngle > PI)
126             botFrontToTargetAngle -= 2 * PI;
127
128         // Decide which way to steer in order to get to the ball.
129         if (abs(botFrontToTargetAngle) > .01) {
130             if (botFrontToTargetAngle > 0) {
131                 controller.steer = 1;
132             }
133             else {
134                 controller.steer = -1;
135             }
136         }
137         else {
138             controller.steer = 0;
139         }
140
141         controller.throttle = 1.0f;
142
143         brain_spec.actions_base().set_source(
144             falken::ActionsBase::kSourceHumanDemonstration);
145
146         brain_spec.actions_base().attribute("steering")->set_joystick_x_axis(
147             controller.steer);
148         brain_spec.actions_base().attribute("steering")->set_joystick_y_axis(0);
149         brain_spec.actions_base().attribute("throttle")->set_number(
150             controller.throttle);
```

```
151     } else {
152         brain_spec.actions_base().set_source(falken::ActionsBase::kSourceNone);
153     }
154
155     episode->Step(0.f);
156
157     if (episode->completed()) {
158         episode = nullptr;
159         Reset();
160         episode = session->StartEpisode();
161     } else if (brain_control) {
162         renderer.DrawString2D("BRAIN CONTROL", rlbot::Color::blue,
163             rlbot::flat::Vector3{ 10, 10, 0 }, 4, 4);
164         float joystick_steer = brain_spec.actions_base().attribute("steering")->
165             joystick_x_axis();
166         if (isnan(joystick_steer)) {
167             controller.steer = 0;
168         } else {
169             controller.steer = joystick_steer;
170         }
171         controller.throttle = brain_spec.actions_base().attribute("throttle")->
172             number();
173     }
174     //renderer.DrawString3D(std::to_string(velocity), rlbot::Color::magenta,
175     //                      ballLocation, 2, 2);
176
177
178     if (gametickpacket->ball()->latestTouch() != NULL) {
179         float new_touch_time = gametickpacket->ball()->latestTouch()->gameSeconds
            ();
180         int new_touch_player = gametickpacket->ball()->latestTouch()->playerIndex
            ();
181         if (!FloatEquals(new_touch_time, last_touch_time) && new_touch_player ==
            index) {
182             std::cout << "I touched it!!";
183             if (brain_control) {
184                 touchCounter += 1;
185                 if (touchCounter > 2) {
186                 switchThreshold += 240;
187                 }
188             }
189             last_touch_time = new_touch_time;
190             Reset();
191             episode->Complete(falken::Episode::kCompletionStateSuccess);
192             episode = session->StartEpisode();
193         }
194     }
195
196     if (gametickpacket->gameInfo()->secondsElapsed() > switchThreshold) {
197         if (!brain_control) {
198             switchThreshold += 30;
199             brain_control = true;
```

```
200             touchCounter = 0;
201         }
202         else {
203             switchThreshold += 20;
204             brain_control = false;
205         }
206     }
207
208     //if (gametickpacket->gameInfo()->secondsElapsed() > snapTime) {
209     //  snapTime += 60 * 15;
210     //  episode->Complete(falken::Episode::kCompletionStateAborted);
211     //  episode = nullptr;
212     //  auto snapshot_id = session->Stop();
213     //  session = nullptr;
214     //  session = brain->StartSession(
215     //      falken::Session::kTypeInteractiveTraining , kMaxSteps);
216     //  Reset();
217     //  //brain_control = true;
218     //  episode = session->StartEpisode();
219     //}
220
221     return controller;
222 }
223
224 bool ExampleBot::InitializeFalken() {
225
226     falken::ObservationsBase observations;
227     falken::EntityBase entity_0(observations, "entity_0");
228
229     falken::ActionsBase actions;
230
231     falken::AttributeBase steering(actions, "steering",
232         falken::kAxesModeDeltaPitchYaw,
233         falken::kControlledEntityPlayer,
234         falken::kControlFrameWorld);
235
236     falken::AttributeBase throttle(actions, "throttle", -1.0f, 1.0f);
237
238     falken::BrainSpecBase brain_spec_base(&observations, &actions);
239
240     static const char* kJsonConfig = nullptr;
241     static const char* project_id = nullptr; /*"MilleniumFalken";*/
242     static const char* api_key = nullptr;
243     service = falken::Service::Connect(
244         project_id, api_key, kJsonConfig);
245
246     static const char* kBrainName = "MilleniumFalken";
247     const char* brain_id = "953716c9-a336-4e4d-a38f-899ee9e4fdb9";
248     const char* snapshot_id = "d9621174-1f29-4473-8990-ddef50ede5f2";
249
250     //brain = service->CreateBrain(kBrainName, brain_spec_base);
251     brain = service->GetBrain(brain_id, snapshot_id);
```

```
252
253     session = brain->StartSession(
254         falken::Session::kTypeInteractiveTraining, kMaxSteps);
255     //session = brain->StartSession(
256     //  falken::Session::kTypeInference, kMaxSteps);
257
258     episode = session->StartEpisode();
259     return true;
260 }
261 bool ExampleBot::FloatEquals(float a, float b) {
262     return (std::abs(a - b) <= absTol * std::max({ 1.0f, std::abs(a), std::abs
      (b) }));
263 }
264
265 void ExampleBot::Reset() {
266     rlbot::GameState gamestate = rlbot::GameState();
267     rlbot::CarState carstate = rlbot::CarState();
268
269     float ball_x = RandomFloat(-3000.f, 3000.f);
270     float ball_y = RandomFloat(-2500.f, 2500.f);
271     float car_x = RandomFloat(-3500.f, 3500.f);
272     float car_y = RandomFloat(-4000.f, 4000.f);
273     float car_yaw = RandomFloat(-PI, PI);
274     float small_v = RandomFloat(-.5f, .5f);
275
276     carstate.physicsState.location = { car_x, car_y, 17.01 };
277     carstate.physicsState.velocity = { 0, 0, 0 };
278     carstate.physicsState.rotation = { 0, car_yaw, 0 };
279     gamestate.ballState.physicsState.location = { ball_x, ball_y, 92.76 };
280     gamestate.ballState.physicsState.rotation = { 0, 0, 0 };
281     gamestate.ballState.physicsState.velocity = { small_v, 0, 0 };
282     gamestate.carStates[index] = carstate;
283
284     rlbot::Interface::SetGameState(gamestate);
285 }
286
287 float ExampleBot::RandomFloat(float Min, float Max) {
288     return ((float(rand()) / float(RAND_MAX)) * (Max - Min)) + Min;
289 }
290
291 void ExampleBot::StartEvaluationSession() {
292     episode->Complete(falken::Episode::kCompletionStateAborted);
293     episode = nullptr;
294     auto snapshot_id = session->Stop();
295     session = nullptr;
296     session = brain->StartSession(falken::Session::kTypeEvaluation, kMaxSteps);
297     Reset();
298     episode = session->StartEpisode();
299     brain_control = true;
300 }
301
302 void ExampleBot::StartInferenceSession() {
```

```
303     // Because the episode was started at Reset, Step the episode before stopping
304     // with an empty episode.
305     episode->Step(0.f);
306     episode->Complete(falken::Episode::kCompletionStateAborted);
307     episode = nullptr;
308     auto snapshot_id = session->Stop();
309     session = nullptr;
310     session = brain->StartSession(falken::Session::kTypeInference, kMaxSteps);
311     Reset();
312     episode = session->StartEpisode();
313 }
```