Team Poggers

CMPUT 350

Build Order

Modified Tank Marine Rush Build

1. Build Supply Depot
2. Build Barrack
3. Build Refinery
4. Build Factory
5. Build Build Starport
6. Build Reactor
7. Build Starport
8. Build Orbital Command
9. Train marines
10. Train Tanks

Scouting

- Use random scv at game start to explore.
- Finds enemy base position and race, and nearby mineral patches.

```
void assignScout() {
    Units SCVs = Observation()->GetUnits(Unit::Alliance::Self, IsUnit(UNIT_TYPEID::TERRAN_SCV));
    if (scouter == NULL) {
        scouter = SCVs.front();
        std::cout << "scout assigned" << std::endl;
    }
}
```

Building Placement

- The given random building placement function was awful, lead to a lot of walking time, and building being unnecessarily spread out.
- Adapted function to base placement near command center.

```
Actions()->UnitCommand(unit_to_build,
                       ability_type_for_structure,
                       Point2D(commandcenter.front()->pos.x + ((rx * depth)+dx),
                           commandcenter.front()->pos.y + ((ry * depth)+dy)));
depth += 0.05f;
```

Early Rush

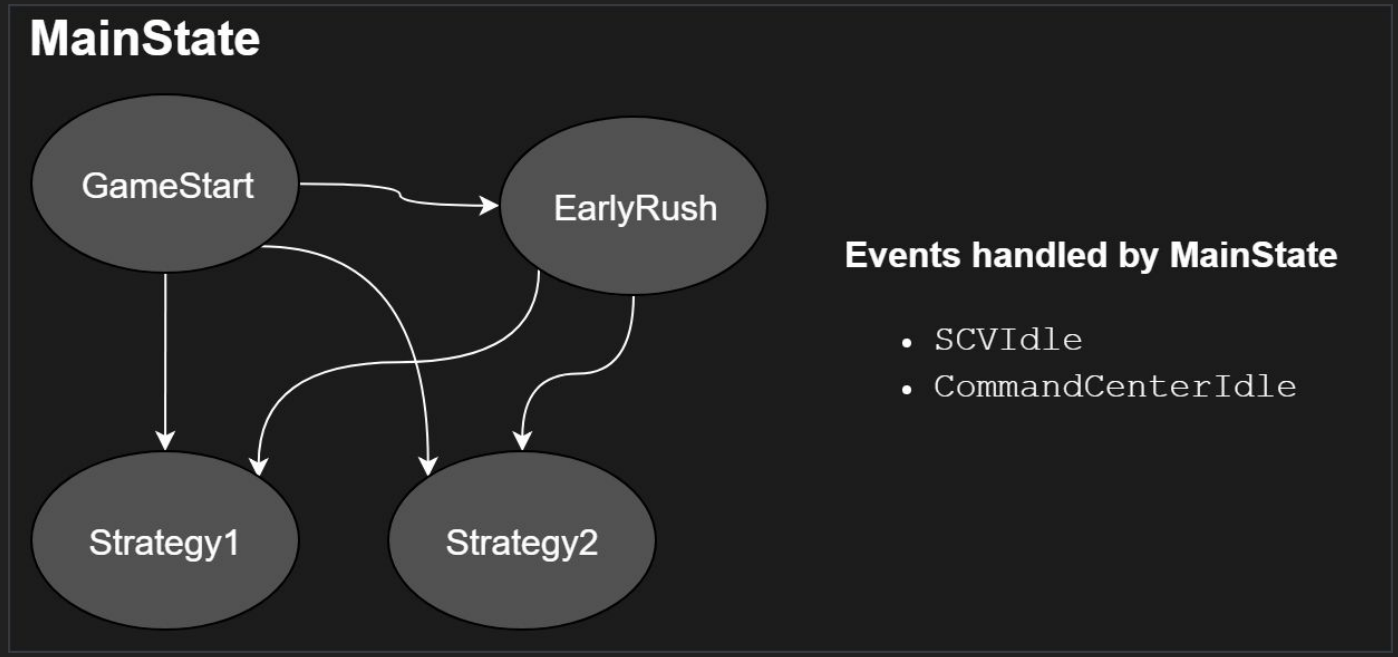
- Use of marines and tanks
- Relatively early, so deals enough damage/stagger
- Intent is to stagger/delay enemy build and harvesters to give us an advantage

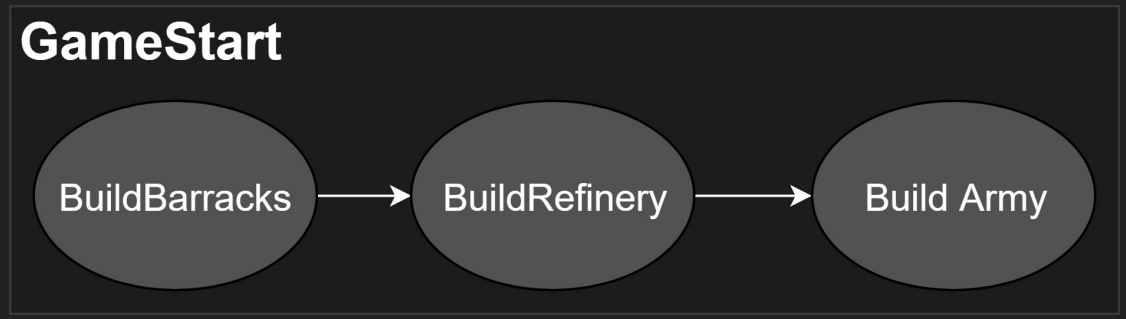
C++ Structure

- Started with lots of if else statements
- Switched to Hierarchical State Machine (HSM)

Boost Statechart

- Hierarchical state machine library using lots of template magic
- Works by sending events to states
  - Ex. StepEvent, UnitCreatedEvent, UnitUpgradedEvent, UnitIdleEvent, etc.
- State processes the event and either stays in current state, or “transits” to another state
- If a state doesn’t have a handler for an event, it passes the event up to its parent
- Each state has its own context that substates can also access


MainState
GameStart
EarlyRush
Strategy1
Strategy2
Events handled by MainState
SCVIdle
CommandCenterIdle


GameStart
BuildBarracks
BuildRefinery
Build Army

Example events

```
struct StepEvent : sc::event<StepEvent> {};
struct UnitCreated : sc::event<UnitCreated> { ... };
struct BuildingConstructed : sc::event<BuildingConstructed> { ... };
struct CommandCenterIdle : sc::event<CommandCenterIdle> { ... };
struct SCVIdle : sc::event<SCVIdle> { ... };
struct BarracksIdle : sc::event<BarracksIdle> { ... };

struct MarineIdle : sc::event<MarineIdle> {
    MarineIdle(const Unit* u) : unit(u) {}
    const Unit* unit;
};

// usage example
state_machine.process_event(MarineIdle(unit));
```

Event handling

- Each state must handle `StepEvent` at minimum
- Can also implement handlers for `BuildingConstructed`, `UnitCreated`, etc. if we wish to override the default handler of a parent state

```
// declaration of a state in its header file
struct GameStart_Refinery : sc::simple_state<GameStart_Refinery, GameStart> {
    GameStart_Refinery() { cout << "GameStart_Refinery state" << endl; }
    // ...
    sc::result react(const StepEvent& event);
    sc::result react(const BuildingConstructed& event);
};
```

Event handling implementation

```
sc::result GameStart_BuildBarracks::react(const StepEvent& event) {
    auto actions = context<StateMachine>().Actions;
    auto observation = context<StateMachine>().Observation;

    if (CountUnitType(observation, UNIT_TYPEID::TERRAN_SUPPLYDEPOT) < 1) {
        TryBuildSupplyDepot(actions, observation);
        return discard_event();
    }
    if (CountUnitType(observation, UNIT_TYPEID::TERRAN_BARRACKS) > 1) {
        return transit<GameStart_Refinery>();
    }

    TryBuildBarracks(actions, observation);
    return discard_event();
}
```

Overriding events

```
sc::result MainState::react(const CommandCenterIdle& e) {
    // only build more SCVs when we have fewer than 16 of them
    auto Observation = context<StateMachine>().Observation;
    if (CountUnitType(Observation, UNIT_TYPEID::TERRAN_SCV) < 16) {
        context<StateMachine>().Actions()->UnitCommand(e.unit, ABILITY_ID::TRAIN_SCV);
    }
    return discard_event();
}

sc::result Strategy2::react(const CommandCenterIdle& e) {
    // this strategy requires more SCVs
    auto Observation = context<StateMachine>().Observation;
    if (CountUnitType(Observation, UNIT_TYPEID::TERRAN_SCV) < 24) {
        context<StateMachine>().Actions()->UnitCommand(e.unit, ABILITY_ID::TRAIN_SCV);
    }
    return discard_event();
}
```

Advantages of HSM

- Break large problem into several smaller states
  - easier to split up work
  - multiple strategies can be implemented simultaneously
- Easy to transition to a different strategy
  - `return transit<DefensiveStrategy>();`
- Can defer generic events to parent states
  - Ex. strategy implementations don’t need to worry about idle worker units
- Each state only stores what it needs
  - no global state means less memory consumption

Downsides of Boost Statechart and HSMs

- Horrible error messages
  - Ordering the structs for states in the wrong order results in >400 errors even when forward declaration is used
- We only implemented the “forward” direction, if we lose lots of units we would have trouble selecting which earlier state to transit to
- Possibility of deadlock if we don’t account for every case

Statistical Evidence

Had testing environment which ran against different races. Ran bot against Easy and Medium AI.

Protoss	Zerg	Terran
Easy: 10/10	Easy: 10/10	Easy: 10/10
Medium: 4/10	Medium: 5/10	Medium: 2/10

Advantages & Disadvantages

+ early push is powerful
+ terrans are strong early
+ disrupts enemy economy
+ especially effective against zerg
- vulnerable after rush
- not good against defensive style
- focused on ground units

Current Issues

- Build orders sometimes incorrect because of placement issues
- Vespene gas farmers not always ideal.
- Second CC lacks proper worker assignment
- Not all units upgraded at proper time
- Orbital scan not utilized

Future

- Implement more strategies and
- Improve ability to detect the strategy of the opponent to counter them
- Gracefully recover from large attacks
- Improve placement of structures to build walls around base
- Remove conflicts in building
- Make build order more consistent
- Fix the way attack is done

Video of Bot

Questions?