캡스톤 디자인의 발표 요령

프로젝트 발표는 10분(데모포함), 질의응답 10분등, 총 20분에 발표 및 데모가 이루어져야 한다. 다음은 캡스톤 디자인 발표 자료에 포함될 내용이며 발표 자료(ppt)의 작성은 최종보고서 및 데모 결과를 토대로 작성한다.

- 프로젝트 제목, 프로젝트 목표 및 개요
- 아키텍처 또는 전체 시스템 구성도 제시
- 주요 기능목록(캡스톤 디자인I의 범위) 및 설명
- 시스템의 성능, 품질 및 제약 요구사항의 테스트 결과
  - 성능 테스트 결과 등
- 주요 알고리즘 및 설계 방법(또는 문제해결 방법)
- 팀원(또는 멘토)의 역할
- 데모

- 질의 응답

캡스톤 디자인 I 최종결과 보고서

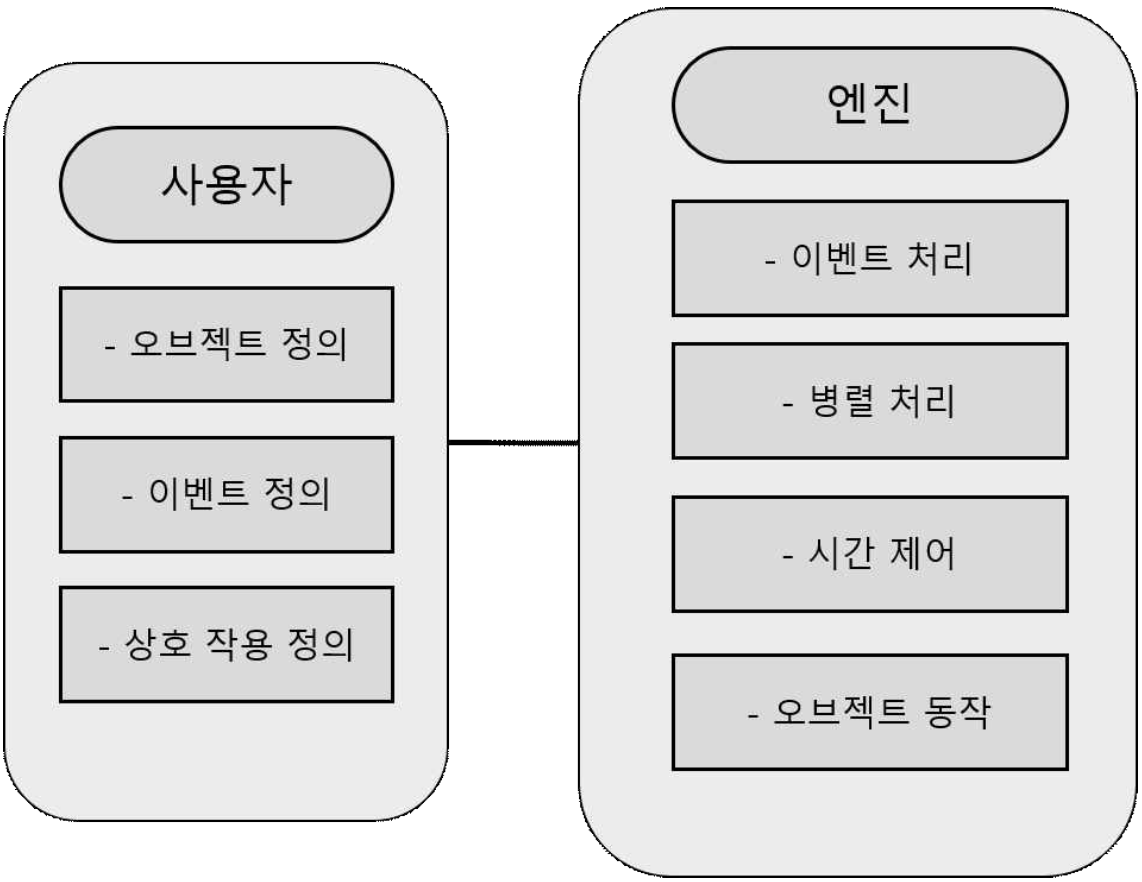
프로젝트 제목(국문): Golang 기반 시뮬레이션 게임 엔진 개발 및 활용
프로젝트 제목(영문): Developing and utilizing Golang-based simulation game engines.

프로젝트 팀(원): 학번: 20171579 이름: 김범수
프로젝트 팀(원): 학번: 20171593 이름: 이제혁
프로젝트 팀(원): 학번: 20171581 이름: 도용주

1. 중간보고서의 검토결과 심사위원의 '수정 및 개선 의견'과 그러한 검토의견을 반영하여 개선한 부분을 명시하시오.

없음

2. 기능, 성능 및 품질 요구사항을 충족하기 위해 본 개발 프로젝트에서 적용한 주요 알고리즘, 설계방법 등을 기술하시오.



본 시뮬레이션 게임 엔진은 에이전트 기반의 이산사건 시뮬레이션 엔진으로
엔진의 사용자는 에이전트를 정의하고 있는 구조체를 임베딩 하고 외부이벤트에 따른 에이전트의 동작함수(Extrnal Transition Function), 외부입력이 없을 때 동작하는 내부동작함수(Internal Transition Function), 내부 동작 함수 호출시 동작하는 출력 함수(Output Function)를 정의하는 인터페이스를 구현 함으로 써 에이전트를 생성 할 수 있다. 추가로 사용자는 에이전트 간의 연결, 내외부 이벤트를 정의하고 에이전트를 엔진에 등록하고 엔진을 동작시킨다.
시뮬레이션/ 게임 엔진에서 중요한 알고리즘은 타임 스케줄링과 시간에 맞추어 에이전트를 동작시키고 상태를 업데이트 해주는 부분이다. 본 프로젝트의 엔진은 사용자가 등록한 에이전트들을 가지고 있다가 요청 시간에 맞추어 생성시킨후 동작시키고, 파괴시간에 맞추어 없앤다. 시간이 흘러감에 따라 내부와 외부에서 발생한 이벤트를 처리하고, 사용자가 정의 한 대로 모델의 입력과 출력을 처리하고 상태를 변화시킨다.

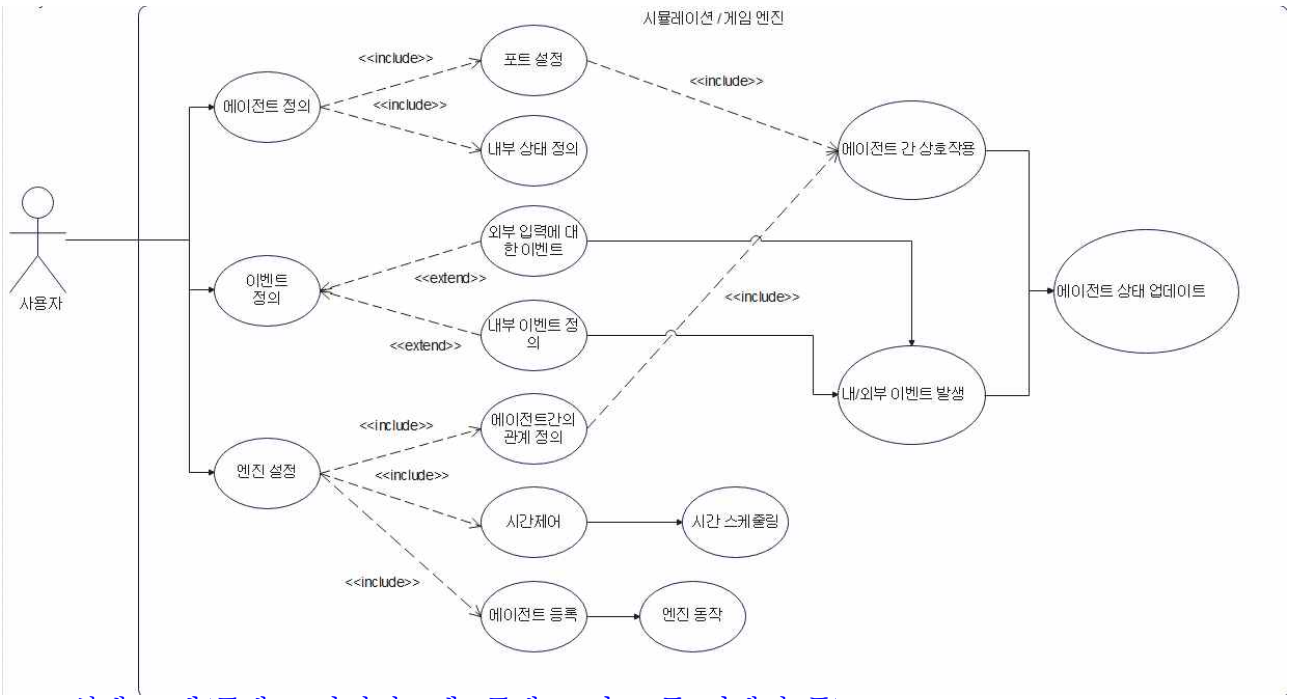
python에서 Golang 으로 포팅 하는 과정에서 두 언어간 차이로 인해 발생하는 다양한 문제가 있었다.

1. python 은 상속을 지원하지만 Golang 에선 상속을 지원하지 않는다.
2. Golang은 추상메서드를 지원하지 않는다.

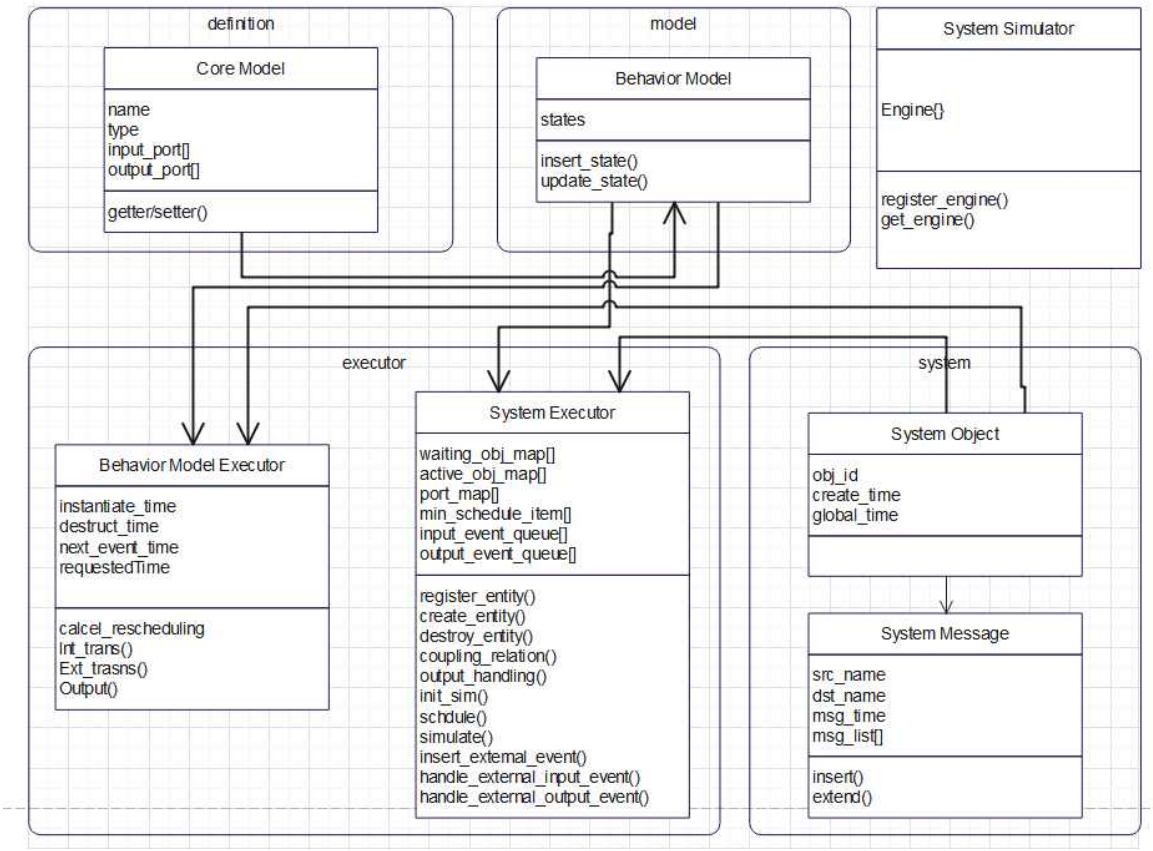
1. 상속의 경우 구조체 임베딩을 통해 상속과 같은 기능을 하도록 했으며,
2. 추상메서드의 경우 인터페이스를 정의하고 인터페이스를 포함하는 구조체를 모델을 구현할 때 임베딩하여 인터페이스를 구현 하도록 했다.

엔진을 Golang으로 포팅하고 성능 개선을 위해 분석하는 과정에서 오브젝트의 수가 많아짐에 따라 오브젝트들을 정렬하는 부분에서 시간이 가장 오래 걸린다는 사실을 알고 이를 해결하기 위해 스케줄링 된 시간에 따라 오브젝트 들을 오름차순으로 가지고 있는 자료구조를 기존 dequeue에서 Golang의 동적배열인 Slice로 바꾸고, Golang의 경량스레드인 Go루틴을 활용해 동시/병렬 적으로 정렬을 수행하는 패키지를 이용해 정렬하도록 했다.

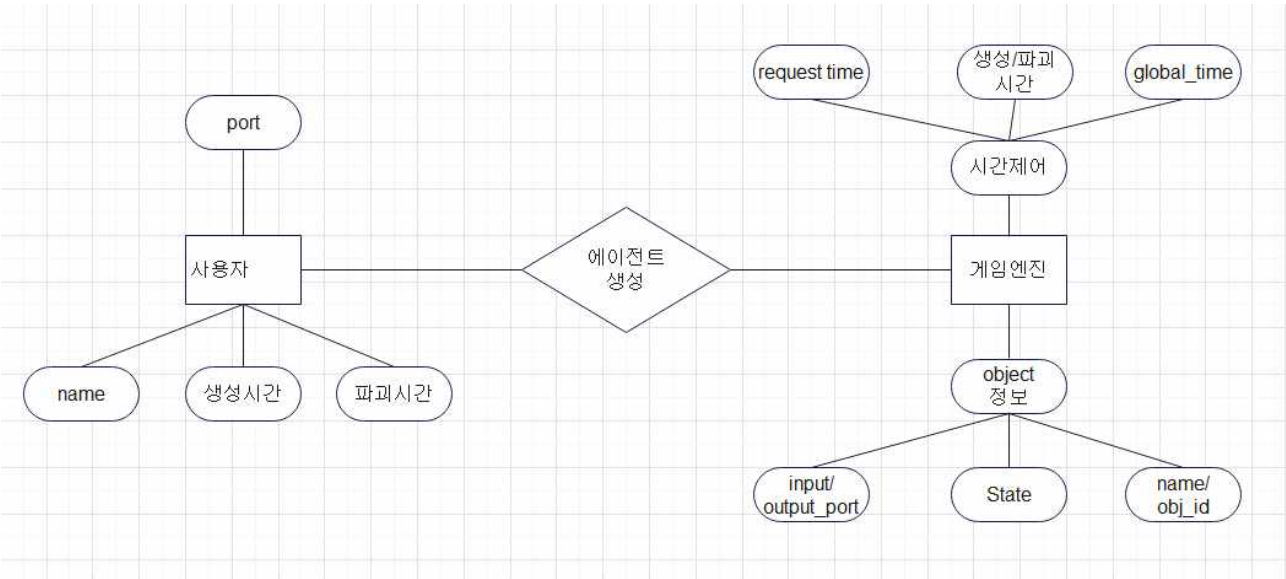
3. 요구사항 정의서에 명세된 기능 및 품질 요구사항에 대하여 최종 완료된 결과를 기술하시오. (작성요령: 최종 완료된 시스템에 대해 아래의 산출물을 작성합니다)
   - 소프트웨어(또는 하드웨어, 시스템) 아키텍처 또는 전체 시스템 구성도
   - 기능별 상세 요구사항(또는 유스케이스)



   - 설계 모델(클래스 다이어그램, 클래스 및 모듈 명세서 등)



- E-R 다이어그램/DB 설계 모델(테이블 구조) // DB를 사용하는 경우에 한함



4. 구현하지 못한 기능 요구사항이 있다면 그 이유와 해결방안을 기술하시오,
(작성요령: 전부 구현한 경우는 "이유"란에 "해당사항 없음"이라고 기재하고, 만약 요구사항대비 구현하지 못한 기능이 있다면 "이유"란에 그 사유를 기재함)

최초 요구사항	구현 여부(미구현, 수정, 삭제 등)	이유(일정부족, 프로젝트 관리미비, 팀원변동, 기술적 문제 등)
병렬 처리	미구현	일정부족

5. 요구사항을 충족시키지 못한 성능, 품질 요구사항이 있다면 그 이유와 해결방안을 기술하시오.

해당사항 없음

6. 최종 완성된 프로젝트 결과물(소프트웨어, 하드웨어 등)을 설치하여 사용하기 위한 사용자 매뉴얼을 작성하시오.

main.go 작성요령

- 패키지 및 모듈 가져오기

```
package main

import (
	"evsim_golang/definition"
	"evsim_golang/executor"
	"evsim_golang/system"
	"fmt"
)
```

- 에이전트 정의

1. 사용할 에이전트 정의

```
type Agent_name struct {
	executor *executor.BehaviorModelExecutor // 에이전트 구조체 임베딩
	// 필요한 구조 정의
}
```

예시)

```
type Generator struct {
	executor *executor.BehaviorModelExecutor
	msg_list []interface{}
}
```

2. 에이전트의 생성자 함수 정의

```
func NewAgnet(instance_time, destruct_time float64, name, engine_name string) *Agent_name {
        agn := Agent_name{} // 에이전트 초기화
        agn.executor = executor.NewExecutor(instance_time, destruct_time, name, engine_name)
        agn.executor.AbstractModel = &agn // 에이전트 인터페이스 대입
        (외부전이함수, 내부전이함수, 출력함수)
        agn.executor.Behaviormodel.Insert_state(STATE1, time_step) // 상태
        agn.executor.Behaviormodel.Insert_state(STATE2, tiem_step)
        agn.executor.Init_state("STATE1") // 상태 초기화
        // 필요한 생성자 정의
        agn.executor.Behaviormodel.CoreModel.Insert_input_port(agn_input_port) // 입력 포트
        agn.executor.Behaviormodel.CoreModel.Insert_output_port(agn_output_port) // 출력 포트
        return &agn
}
```

예시)

```
func NewGenerator(instance_time, destruct_time float64, name, engine_name string) *Generator {
        gen := Generator{}
        gen.executor = executor.NewExecutor(instance_time, destruct_time, name, engine_name)
        gen.executor.AbstractModel = &gen
        gen.executor.Init_state("IDLE")
        gen.executor.Behaviormodel.Insert_state("IDLE", definition.Infinite)
        gen.executor.Behaviormodel.Insert_state("MOVE", 1)
        gen.executor.Behaviormodel.CoreModel.Insert_input_port("start")
        gen.executor.Behaviormodel.CoreModel.Insert_output_port("process")
        for i := 0; i < 10; i++ {
                gen.msg_list = append(gen.msg_list, i)
        }
        return &gen
}
```

3. 외부이벤트에 따른 에이전트의 동작함수 정의

```
func (a *Agent_name) Ext_trans(port string, msg *system.SysMessage) {
        // 외부이벤트에 따른 에이전트의 동작
}
```

예시)

```
func (g *Generator) Ext_trans(port string, msg *system.SysMessage) {
        if port == "start" {
                g.executor.Cur_state = "MOVE"
        }
}
```

4. 내부 동작 함수 호출시 동작하는 출력 함수 정의

```
func (a *Agent_name) Output() *system.SysMessage {
        // 출력 함수 정의
}
```

예시)

```
func (g *Generator) Output() *system.SysMessage {
        fmt.Println(g.msg_list...)
        g.msg_list = remove(g.msg_list, 0)
        return nil
}
```

5. 외부입력이 없을 때 동작하는 내부동작함수 정의

```
func (a *Agent_name) Int_trans() {
        // 내부동작함수 정의
}
```

예시)

```
func (g *Generator) Int_trans() {
        if g.executor.Cur_state == "MOVE" && len(g.msg_list) == 0 {
                g.executor.Cur_state = "IDLE"
        } else {
                g.executor.Cur_state = "MOVE"
        }
```

```
}
```

- 메인 정의

```
func main() {
        se := executor.NewSysSimulator()                    // 엔진 생성
        se.Register_engine(engine_name, sim_mode, time_step) // 엔진 등록
        sim := se.Get_engine(engine_name)                    // 엔진 가져오기

        sim.Behaviormodel.CoreModel.Insert_input_port(input_port) // 입력 포트 생성

        agn := NewGenerator(instance_time, destruct_time, name, engine_name) // 에이전트 가져오기
        // 오브젝트의 생성시간, 파괴시간, 이름, 등록할 엔진이름

        agn.Register_entity(agn.executor) // 에이전트 등록

        agn.Coupling_relation(nil, input_port, agn.executor, agn_input_port)
        // 엔진 포트와 오브젝트 포트 연결

        agn.Insert_external_event(port, msg, req_time) // 발생시킬 이벤트 정의

        agn.Simulate(definition.Infinite) // 엔진 실행
}
```

예시)

```
func main() {
        se := executor.NewSysSimulator()
        se.Register_engine("sname", "REAL_TIME", 1)
        sim := se.Get_engine("sname")

        sim.Behaviormodel.CoreModel.Insert_input_port("start")

        gen := NewGenerator(0, definition.Infinite, "Gen", "sname")

        sim.Register_entity(gen.executor)

        sim.Coupling_relation(nil, "start", gen.executor, "start")

        sim.Insert_external_event("start", nil, 0)
```

```
        sim.Simulate(definition.Infinite)
}
```

7. 캡스톤디자인 결과의 활용방안

시뮬레이션은 현실에서 일어나는 현상과 상황을 분석하고 가상현실에 반영해 구동하여 얻어진 데이터를 바탕으로 위험에 대응하거나, 효율적인 운영에 활용되거나, 다양한 문제 해결에 기여할 수 있다. 그렇기 때문에 시뮬레이션 / 게임 엔진은 재난대피, 국방, 도로교통체계 개발 등 여러 분야에서 의사결정 과 정책결정을 하는데 활용되고 있다. 현재 시대가 빠르게 변화하는 만큼 그에 발 맞추어 컨텐츠를 개발하고, 빠른 의사결정과 정책결정이 중요해짐에 따라 시뮬레이션의 결과를 빠르게 도출하는 것 역시 중요해지고 있다. 그리고 기술이 발전함에 따라 시뮬레이션의 범위도 광대해졌고 시뮬레이션/게임 엔진은 수많은 오브젝트들을 생성하고 처리해야 한다. 본 캡스톤 디자인의 결과로 완성된 프로젝트는 에이전트 기반의 이산사건 시뮬레이션 엔진으로 사용자가 자신의 의도에 맞게 에이전트를 정의하고, 에이전트를 구현하여 시뮬레이션을 수행하게끔 하는데, 많은 수의 에이전트를 생성해도 빠른 성능을 제공하여 사용자가 빠른 시뮬레이션의 결과를 도출하도록 도와 의사결정과 정책결정 을 하는데있어 활용하거나, MMORPG와 같이 광범위한 프로젝트에 활용되어 사용자에게 뛰어난 성능을 제공할 수 있다.