第四课课程笔记

概念

Dojo的核心组成部分

1. ECS (entity component system)
2. Torii
3. Katana
4. Sozo CLI

Autonomous World(自治世界)

1. 持久性和去中心化
2. 开放参与
3. 去中心化的数据可用性层
4. 无需许可的扩展

搭建环境与基本命令

1. 安装Dojoup
   `curl -L https://install.dojoengine.org | bash`
2. 安装Torii, Katana, Sozo
   `dojoup`
3. 新建项目
   `sozo init`
4. 编译
   `sozo build`
5. 运行 Katana/
   `katana --disable-fee`
6. 部署到 Katana
   `sozo migrate --name test`
7. 打开torii索引 // 复制 sozo migrate 输出的 world address
   `torii --world <WORLD_ADDRESS>`
8. 与action交互 // 复制 sozo migrate 输出的 action address

```
sozo execute <SYSTEM_ACTION_ADDRESS> spawn
sozo execute <SYSTEM_ACTION_ADDRESS> move -c 1
```

9. 查询访问最新状态 `http://0.0.0.0:8080/graphql`

```
query {
  movesModels {
    edges {
      node {
        player
        remaining
        last_direction
      }
```

```
    }
  },
  positionModels {
    edges {
      node {
        vec {
          x,
          y
        }
      }
    }
  }
}
```

项目解析

```
.
├── LICENSE
├── README.md
├── Scarb.lock
├── Scarb.toml
├── assets
│   ├── cover.png
│   └── icon.png
├── scripts
│   └── default_auth.sh
├── src
│   ├── lib.cairo
│   ├── models
│   │   ├── moves.cairo
│   │   └── position.cairo
│   ├── systems
│   │   └── actions.cairo
│   └── tests
│       └── test_world.cairo
└── target
    ├── CACHEDIR.TAG
    └── dev
        ├── dojo::base::base.json
        ├── dojo::executor::executor.json
        ├── dojo::world::world.json
        ├── dojo_starter::models::moves::moves.json
        ├── dojo_starter::models::position::position.json
        ├── dojo_starter::systems::actions::actions.json
        └── manifest.json
```

models（代表components, 组件编写）

`Into` `#[key]` `Introspect`

systems（游戏逻辑实现）

`get!` `set!` `emit!`

tests（components和system的测试）

```
#[cfg(test)]
```

```
mod tests {
    use core::debug::PrintTrait;
    use core::option::OptionTrait;
    use core::traits::TryInto;
    use dojo_starter::models::position::Vec2Trait;
    use core::traits::Into;
    use starknet::class_hash::Felt252TryIntoClassHash;

    // import world dispatcher
    use dojo::world::{IWorldDispatcher, IWorldDispatcherTrait}; //system

    // import test utils
    use dojo::test_utils::{spawn_test_world, deploy_contract}; // system

    // import test utils
    use dojo_starter::{
        systems::{actions::{actions, IActionsDispatcher,
IActionsDispatcherTrait}},
        models::{position::{Position, Vec2, position}, moves::{Moves, Direction,
moves}}
    };

    ///////////////   models  test
    #[test]
    #[available_gas(300000000000)]
    fn test_direction_into() {
        let dir = Direction::Left;
        assert(dir.into() == 1, 'NOT 1');
    }

    #[test]
    #[available_gas(2000000)]
    fn test_vec_is_zero() {
        let vec2 = Vec2 { x: 1, y: 1 };
        assert(vec2.is_zero(), 'VEC NOT 0');
    }

    #[test]
    #[available_gas(2000000)]
    fn test_vec_is_equal() {
        let vec2_1 = Vec2 { x: 100, y: 100 };
        let vec2_2 = Vec2 { x: 100, y: 100 };
        assert(vec2_1.is_equal(vec2_2), '1 should equal 2');
    }

    ///////////////   systems  test
    #[test]
    #[available_gas(2000000000000)]
    fn test_spawn() {
        let caller = starknet::contract_address_const::<0x0>();
        let mut models = array![position::TEST_CLASS_HASH,
moves::TEST_CLASS_HASH];
        let world = spawn_test_world(models);

        let contract_address = world
```

```
            .deploy_contract('salt',
actions::TEST_CLASS_HASH.try_into().unwrap());

        let actions_system = IActionsDispatcher { contract_address };

        actions_system.spawn();

        let (moves, positions) = get!(world, caller, (Moves, Position));

        assert(moves.remaining == 100, 'not 100');
        assert(moves.last_direction.into() == 0, 'not 0');

        assert(positions.vec.x == 10, 'not 10');
        assert(positions.vec.y == 10, 'not 10');
    }


    #[test]
    #[available_gas(30000000)]
    fn test_move() {
        // caller
        let caller = starknet::contract_address_const::<0x0>();

        // models
        let mut models = array![position::TEST_CLASS_HASH,
moves::TEST_CLASS_HASH];

        // deploy world with models
        let world = spawn_test_world(models);

        // deploy systems contract
        let contract_address = world
            .deploy_contract('salt',
actions::TEST_CLASS_HASH.try_into().unwrap());
        let actions_system = IActionsDispatcher { contract_address };

        // call spawn()
        actions_system.spawn();

        // call move with direction right
        actions_system.move(Direction::Right);

        // Check world state
        let moves = get!(world, caller, Moves);

        // casting right direction
        let right_dir_felt: felt252 = Direction::Right.into();

        // check moves
        assert(moves.remaining == 99, 'moves is wrong');

        // check last direction
        assert(moves.last_direction.into() == right_dir_felt, 'last direction is
wrong');
```

```
        // get new_position
        let new_position = get!(world, caller, Position);

        // check new position x
        assert(new_position.vec.x == 11, 'position x is wrong');

        // check new position y
        assert(new_position.vec.y == 10, 'position y is wrong');
    }
}
```

参考资料

https://hackmd.io/@wongssh/dojo
https://book.dojoengine.org/
https://www.veradiverdict.com/p/dojo?r=cfpxn&utm_campaign=post&utm_medium=email
https://github.com/keep-starknet-strange/tsubasa