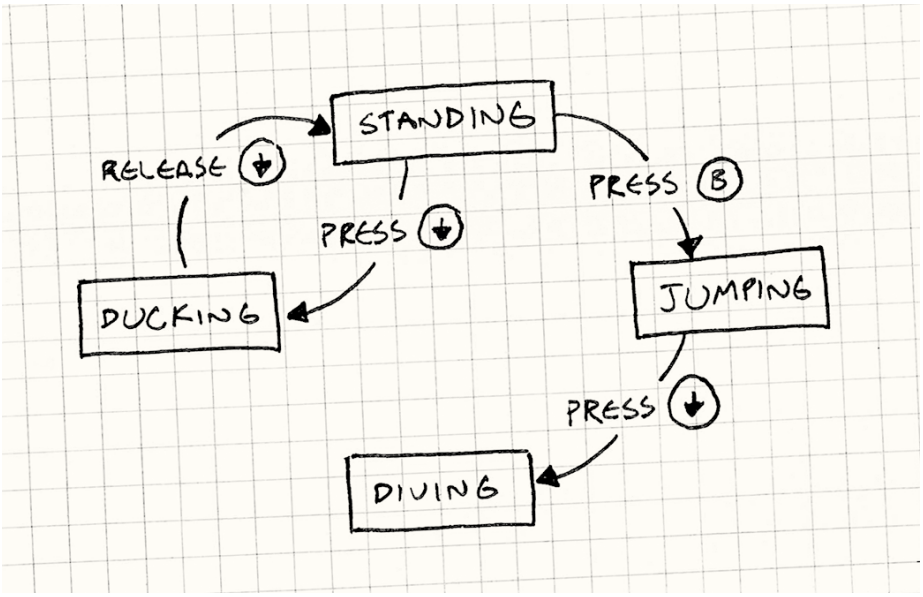
Game Design Patterns Website

States

Finite State Machines



The gist of this is: * You have a fixed set of states that the machine can be in. For example, standing, jumping, ducking, and diving. * The machine can only be in one state at a time. * A sequence of inputs or events is sent to the machine. * Each state has a set of transitions, each associated with an input and pointing to a state.

A State Interface

First, we define an interface for the state. Everything that is state-dependent becomes a virtual method in that interface. For example if you creating an interface for a hero:

```
class HeroineState
{
public:
    virtual ~HeroineState() {}
    virtual void handleInput(Heroine& heroine, Input input) {}
    virtual void update(Heroine& heroine) {}
};
```

Classes for Each State

For each state, we define a class that implements the interface.

```
class DuckingState : public HeroineState
{
public:
    DuckingState():
        chargeTime_(0)
    {}

    virtual void handleInput(Heorine& heroine, Input input)
    {
        // Change to standing state...
        if (input == RELEASE_DOWN)
            heroine.setGraphics(IMAGE_STAND);
        }
    }

    virtual void update(Heroine& heroine)
    {
        chargeTime_++;
        if (chargeTime_ > MAX_CHARGE)
            heroine_superBomb();
        }
};
```

Delegate To The State

Next, we give the Heroine a pointer to her current state.

```
class Heroine
{
public:
    virtual void handleInput(Input input)
    {
    state_->handleInput(*this, input);
    }

    virtual void update()
    {
    state_->update(*this);
    }

    // Other methods...
private:
    HeroineState* state_;
};
```