A STEP TOWARDS DATA ORIENTATION

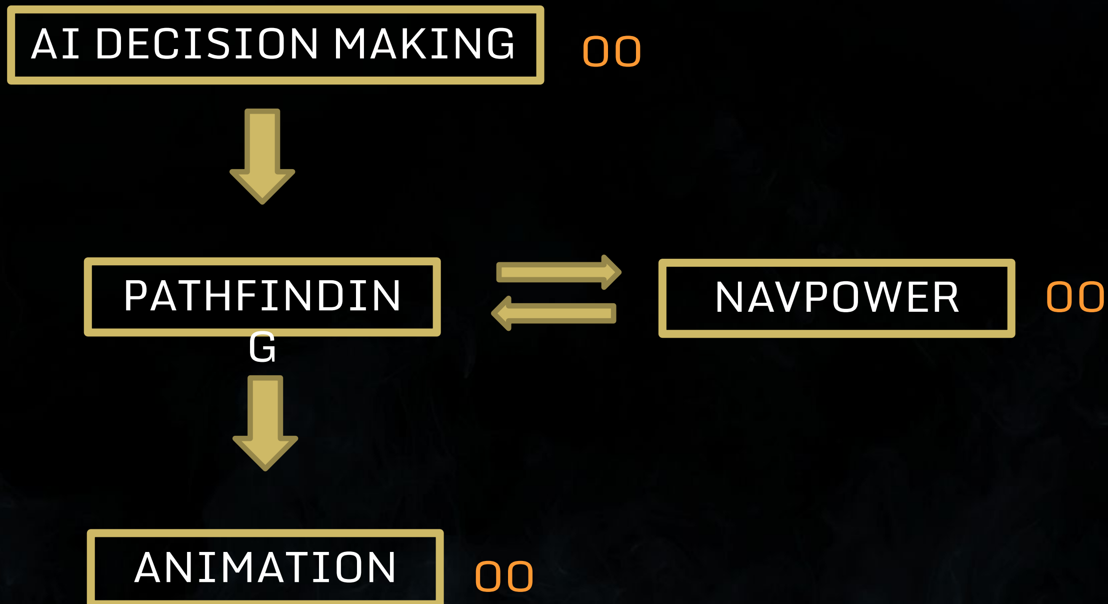
JOHAN TORP
<JOHAN.TORP@DICE.SE>

DICE CODERS DAY 3/11 2010

AGENDA

- AI decision making - Pathfinding - Animation
- Look at actual code & patterns
- Questions

IN AN OO WORLD



DICE
FROSTBITE
A DICE TECHNOLOGY

DECISION TO MOVEMENT

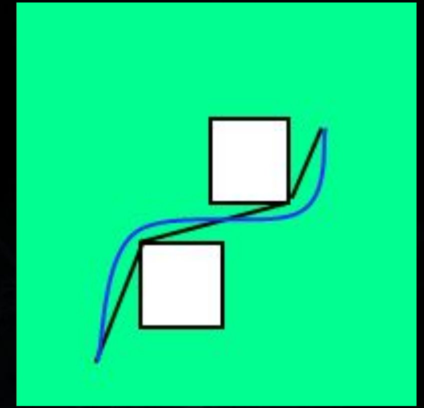
DECISION TO MOVEMENT

DICE

FROSTBITE
A DICE TECHNOLOGY

DECISION TO MOVEMENT

DECISION TO MOVEMENT



FROSTBITE
A DICE TECHNOLOGY

DECISION TO MOVEMENT

NAVPOWER OPERATIONS

- Find path
- Load / unload nav mesh section
- Add / remove obstacles
- Path invalidation detection
- Can go-tests
- “Raycasts”, circle tests, triangle tests

FROSTBITE
A DICE TECHNOLOGY

ABSTRACTS

- Pathfinder - find path, path invalidation, circle tests, raycasts
- Random position generator - can go-tests
- Manager - load nav mesh, obstacles, destruction, updates

Left some raycasts synchronous

PATHFINDER INTERFACE

```
interface Pathfinder
{
public:
  virtual PathHandle* findPath(const PathfindingPosition& start,
                               const PathfindingPosition& end,
                    Optional<float> corridorRadius,
                    PathHandle::StateListener* listener) = 0;

  /// More efficient version of findPath when start is the end of a previous path
  ///
  /// @pre lastPath->getState() == PathHandle::ValidPath
  virtual PathHandle* findPathFromDestination(PathHandle* lastPath,
                                       const PathfindingPosition& end,
                          Optional<float> corridorRadius,
                          PathHandle::StateListener* listener) = 0;

  virtual void releasePath(PathHandle* path) = 0;

  virtual bool canGoStraight(Vec3Ref start, Vec3Ref end, Vec3* collision = nullptr) const = 0;
};
```

FROSTBITE
A DICE TECHNOLOGY

PATH HANDLE

```
typedef fixed_vector<Vec3, 16> WaypointVector;
typedef fixed_vector<float, 16> WaypointRadiusVector;

struct PathHandle
{
  enum State {ComputingPath, ValidPath, NoPathAvailable, RepathingRequired};

  interface StateListener {
        virtual void onStateChanged(PathHandle* handle) = 0;
  };

  PathHandle() : waypoints(pathfindingArena()), radii(pathfindingArena()) {}

  WaypointVector waypoints;
  WaypointRadiusVector radii;

  State state;
};
```

PATH HANDLE

```
typedef eastl::fixed_vector<Vec3, 16> WaypointVector;
typedef eastl::fixed_vector<float, 16> WaypointRadiusVector;

struct PathHandle
{
  enum State {ComputingPath, ValidPath, NoPathAvailable, RepathingRequired};

  interface StateListener {
       virtual void onStateChanged(PathHandle* handle) = 0;
  };

  PathHandle() : waypoints(pathfindingArena()), radii(pathfindingArena()) {}

  WaypointVector waypoints;
  WaypointRadiusVector radii;

  State state;
};
```

DICE

FROSTBITE
A DICE TECHNOLOGY

NAVPOWER PATHFINDER

```
class NavPowerPathfinder : public Pathfinder
{
public:
    virtual PathHandle* findPath(...) override;
    virtual PathHandle* findPathFromDestination(...) override;
    virtual void releasePath(...) override;
    virtual bool canGoStraight(...) const override;

    void updatePaths();
    void notifyPathListeners();

private:
    bfx::PolylinePathRCPtr m_paths[MaxPaths];
    PathHandle m_pathHandles[MaxPaths];
    PathHandle::StateListener* m_pathHandleListeners[MaxPaths];
    u64 m_usedPaths;

    typedef eastl::fixed_vector<PathHandle*, MaxPaths, false> PathHandleVector;
    PathHandleVector m_updatedPaths, m_updatedValidPaths;
};
```

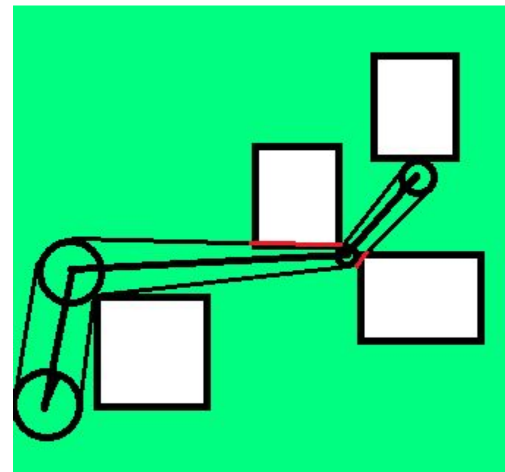
CORRIDOR STEP

```
› typedef eastl::vector<CorridorNode> Corridor;

› ScratchPadArena scratch;
› Corridor corridor(scratch);
› corridor.resize(navPowerPath.size()); // Will allocate memory using the scratch pad
```

1. Copy all new NavPower paths -> temporary representation
2. Drop unnecessary points
3. Corridor adjust paths who requested it
4. Copy temporaries -> PathHandles

CORRIDOR STEP 2-4

```
›   const CorridorHandleVector::iterator allBegin = all.begin(), allEnd = all.end();
›   const CorridorHandlePtrVector::iterator adjustBegin = adjust.begin(), adjustEnd = adjust.end();

›   for (CorridorHandleVector::iterator it=allBegin; it!=allEnd; ++it)
›     dropUnnecessaryPoints(it->corridor, scratchPad);

›   for (CorridorHandlePtrVector::iterator it=adjustBegin; it!=adjustEnd; ++it)
›     shrinkEndPoints((**it).corridor, m_id);

›   for (CorridorHandlePtrVector::iterator it=adjustBegin; it!=adjustEnd; ++it)
›     calculateCornerDisplacements((**it).corridor);

›   for (CorridorHandlePtrVector::iterator it=adjustBegin; it!=adjustEnd; ++it)
›     displaceCorners((**it).corridor, m_id);

›   for (CorridorHandlePtrVector::iterator it=adjustBegin; it!=adjustEnd; ++it)
›     shrinkSections((**it).corridor, m_id);

›   for (CorridorHandleVector::iterator it=allBegin; it!=allEnd; ++it)
›     copyCorridorToHandle(it->corridor, *it->handle);
}
```



DICE
FROSTBITE
A DICE TECHNOLOGY

NAVPOWER MANAGER

```
› void NavPowerManager::update(float frameTime)
› {
          m_streamingManager.update();

          m_destructionManager.update();

›    m_obstacleManager.update();

›    bfx::SystemSimulate( frameTime );

›    for (PathfinderVector::const_iterator it=m_pathfinders.begin(), ...)
   ›           (**it).updatePaths();

›    for (PathfinderVector::const_iterator it=m_pathfinders.begin(), ...)
   ›           (**it).notifyPathListeners();

›    for (PositionGeneratorVector::const_iterator it=m_positionGenerators.begin(), end = ...)
›             (**it).update();
› }
```

DICE

FROSTBITE
A DICE TECHNOLOGY

BATCHING BENEFITS

- Keep pathfinding code/data cache hot
- Avoid call sites cache running cold
- Easier to jobify / SPUify
- Easy to timeslice

ASYNCHRONOUS

- Manager
- Random position generator
- Pathfinder

Collect destruction messages, process in batch

Runs ~1/sec. Allows synchronous decisions

Decision making assumes success

LESS SIMPLIFIED ARCHITECTURE

SCRIPTING

AI DECISION MAKING

PATH FOLLOWING

PATHFINDING

NAVPOWER

DRIVING

LOCOMOTION

VEHICLE INPUT

SERVER

CLIENT

LOCOMOTION

ANIMATION

Waypoints

Waypoint Data

Corridor Radii

MY PRECIOUS LATENCY

Each server tick

1. Each AI decision making
2. Pathfinding manager update
   - All pathfinding requests
   - All corridor adjustments
   - All PathHandle notifications -> path following -> server locomotion
3. Network pulse. Server locomotion -> client locomotion
4. ...rest of tick

ASYNCHRONOUS EXAMPLES

- Callbacks. Delay? Fire in batch?
- Handle+poll instead of callbacks. Poll in batch.
- Record messages, process all once a frame
- Check success / failure next frame
- Pre-calculate what you're likely to need

Con: Callstack won't tell you everything.
...but can we afford deep callstacks?

RESOLVE EARLY

```
void Bot::changeVehicle(const ServerEntryComponent* entry)
{
    ...

    m_pathFollower = entry->owner()->
        getFirstComponentOfType<ServerPathFollowingComponent>()->getPathFollower();
}
```

DICE

FROSTBITE
A DICE TECHNOLOGY

BE NICE TO YOUR MEMORY ☺️

new / push_back() / insert() / resize()

Stop and think!

- Where is the memory allocated?
- Pre-allocated containers?
- Scratch pad?
- Can I resize()/reserve() immediately?
- Can I use Optional<T> instead of ScopedPtr<T>?
- Can I use vectors instead of list / set / map?

FROSTBITE

A DICE TECHNOLOGY

LET'S START MOVING

- Let's not abandon OO nor rewrite the world
- Start small, batch a bit, resolve inputs, avoid deep dives, grow from there
- Much easer to rewrite a system in a DO fashion afterwards

SUMMARY

AI decision making – pathfinding – animation
Code: Handles, arena, scratch pad, fixed_vector, batch processing
Latency analysis, async patterns
Think about depth/width of calls, try stay within your system, resolve early,
new/push_back() = think

QUESTIONS?

- johan.torp@dice.se