Motivation

- The last thing you want to do is write critical code near the end of a project
  - Induces huge stress on the team
  - Introduces all kinds of interesting bugs that break working code
- Testing always gets cut in a crunch
  - Makes the problem even worse!
- Planning can help avoid writing critical code in alpha or beta phases
Wishes Versus Reality

- Most games you play are less/smaller than originally envisioned
  - Design was bigger than implementation
  - Implementation was bigger than what actually made it into the game

- How do we know when a game is "done"?

How Do We Estimate Progress?

- Example:
  - Jo is a programmer
  - She estimates it will take 10 days to implement a Smart Trap
  - She is 4 days into the implementation
  - Is the Smart Trap 40% complete?
    - We may not see it "snap shut" until day 9
    - Say she is good, and finishes in 8 days total
      - We are ahead!
  - Later, it is decided to add functionality to the Smart Trap (e.g., can trap larger objects)
    - This takes 4 days
    - Now we’re behind!
So, What’s the Point?

- Most things get revisited multiple times during development
  - Fix bugs, modify functionality, etc.
- The "40% done" estimate looks pretty sketchy...
- We need a way to account for time without driving a project into trouble (and into panic)

Incremental Delivery

- Milestones are good things!
  - They let us get things done
- Downside
  - If you miss one, people notice, and action is often taken
  - Especially management and production people
Incremental Delivery (cont.)

- **Developer’s view**
  - Milestones (or plans in general) are just *best guesses* for how the implementation will evolve

- **Management’s view**
  - Schedules are contracts with developers
  - Promising certain things at certain times

- These different views cause problems
  - Developers: Panic, pressure, long hours
  - Managers: Justification, financial pressure

Milestones

- Without milestones, work will not get done

- **Unrealistic** milestones mean the work will not get done on time, regardless of how financially important they are

- Managers need to know the estimates of the developers, and the key markers along the way
  - They need to plan their financial links accordingly
Milestones (cont.)

- External (used by managers) milestones are at a coarser granularity
  - Need to tie to publishers, etc.

- Internal (used by developers) milestones are at a finer granularity
  - Need to use among team members

Think of the development plan as a blackbox

- Managers have a specific "interface" to the box
  - Give me the latest build
  - Give me the latest (high-level) schedule

Clearly, this is too simplistic/wishful thinking

- Managers want to know more

But it helps separate things better
Hidden Gems

- For many, if I can’t see it, it is not important
  - AI takes time to build
  - Network balancing is an optimization
- Developers receive less "credit" for these than things that can be seen
- Good managers will probe deeper below the surface to see what is really going on
  - Requires technical ability (knowledge)

Iteration

- Make frequent (daily, weekly?) working builds
  - "We don’t go home Friday until a working build is checked in."
  - If management asks for the latest build, give them the one from last week
- Resist the desire to show the latest-and-greatest
  - People will always expect it, and it leads to unrealistic expectations
Internal Scheduling

- Given a detailed design document
  - Make a list of all objects (players, items, NPCs, environments, etc.) that need to be built
  - Mark each one as either
    - Core,
    - Required, or
    - Desired.
  - Remember the circle diagram?

- End result
  - List of features sorted by importance

Internal Scheduling (cont.)

- Could start working from top of list, and when time runs out, we are done
  - Produces a lot of complete pieces, but no whole
  - Makes management (and others) nervous

- Since we made the list in an OO way, we should start building objects!
OO Iterative Development: Object Versions

- Create a *Null* version for each object
  - Complete, but empty

- *Basic* version
  - Placeholder with some properties present

- *Nominal* version
  - Commercially viable implementation

- *Optimal* version
  - State of the art version

```cpp
// Player.h
class Player {
    public:
        Player( void );
        ~Player( void );
};

//Player.cpp
#include "Player.h"
Player::Player( void ) {
}
Player::~Player( void ) {
}
```

OO Iterative Development: Object Versions (cont.)

- Some objects will be simpler
  - Fewer iterations

- Some will be more complex
  - More iterations

- We can say we have a *shippable* game when every object is at least at the *Nominal* version

- A *complete* game is one where all objects are at *Optimal* level
Discussion

☐ Seems like we need to write three versions of every object!
  ■ Yes, but we would probably do this anyway with revisions

☐ Approach
  ■ Starting with core, then required, then desired, implement Null versions of all objects
  ■ Starting with core, then required, implement the Nominal versions
     ☐ Code is now releasable
  ■ Start to work on desirables

Discussion (cont.)

☐ This is a breadth-first approach

☐ Better than "let's do the cool bits first!"
  ■ Always have a build-able game
  ■ Near-continuous growth
  ■ Can easily show refinement
  ■ Better handle on how "complete" the game is
## Scheduling: Naïve

<table>
<thead>
<tr>
<th>Feature</th>
<th>Null</th>
<th>Base</th>
<th>Nominal</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>1</td>
<td>13</td>
<td>25</td>
<td>37</td>
</tr>
<tr>
<td>F2</td>
<td>2</td>
<td>14</td>
<td>26</td>
<td>38</td>
</tr>
<tr>
<td>F3</td>
<td>3</td>
<td>15</td>
<td>27</td>
<td>39</td>
</tr>
<tr>
<td>F4</td>
<td>4</td>
<td>16</td>
<td>28</td>
<td>40</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td>5</td>
<td>17</td>
<td>29</td>
<td>41</td>
</tr>
<tr>
<td>F6</td>
<td>6</td>
<td>18</td>
<td>30</td>
<td>42</td>
</tr>
<tr>
<td>F7</td>
<td>7</td>
<td>19</td>
<td>31</td>
<td>43</td>
</tr>
<tr>
<td>F8</td>
<td>8</td>
<td>20</td>
<td>32</td>
<td>44</td>
</tr>
<tr>
<td>Desired</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F9</td>
<td>9</td>
<td>21</td>
<td>33</td>
<td>45</td>
</tr>
<tr>
<td>F10</td>
<td>10</td>
<td>22</td>
<td>34</td>
<td>46</td>
</tr>
<tr>
<td>F11</td>
<td>11</td>
<td>23</td>
<td>35</td>
<td>47</td>
</tr>
<tr>
<td>F12</td>
<td>12</td>
<td>24</td>
<td>36</td>
<td>48</td>
</tr>
</tbody>
</table>

## Scheduling: Better (single programmer)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Null</th>
<th>Base</th>
<th>Nominal</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>1</td>
<td>13</td>
<td>22</td>
<td>37</td>
</tr>
<tr>
<td>F2</td>
<td>2</td>
<td>14</td>
<td>23</td>
<td>38</td>
</tr>
<tr>
<td>F3</td>
<td>3</td>
<td>15</td>
<td>24</td>
<td>39</td>
</tr>
<tr>
<td>F4</td>
<td>4</td>
<td>16</td>
<td>25</td>
<td>40</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td>5</td>
<td>17</td>
<td>26</td>
<td>41</td>
</tr>
<tr>
<td>F6</td>
<td>6</td>
<td>18</td>
<td>27</td>
<td>42</td>
</tr>
<tr>
<td>F7</td>
<td>7</td>
<td>19</td>
<td>28</td>
<td>43</td>
</tr>
<tr>
<td>F8</td>
<td>8</td>
<td>20</td>
<td>29</td>
<td>44</td>
</tr>
<tr>
<td>Desired</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F9</td>
<td>9</td>
<td>21</td>
<td>32</td>
<td>45</td>
</tr>
<tr>
<td>F10</td>
<td>10</td>
<td>30</td>
<td>33</td>
<td>46</td>
</tr>
<tr>
<td>F11</td>
<td>11</td>
<td>31</td>
<td>34</td>
<td>47</td>
</tr>
<tr>
<td>F12</td>
<td>12</td>
<td>35</td>
<td>36</td>
<td>48</td>
</tr>
</tbody>
</table>
Scheduling:
Better (multiple programmers)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Null</th>
<th>Base</th>
<th>Nominal</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>1A</td>
<td>7A</td>
<td>11B</td>
<td>19A</td>
</tr>
<tr>
<td>F2</td>
<td>1B</td>
<td>7B</td>
<td>12A</td>
<td>19B</td>
</tr>
<tr>
<td>F3</td>
<td>2A</td>
<td>8A</td>
<td>12B</td>
<td>20A</td>
</tr>
<tr>
<td>F4</td>
<td>2B</td>
<td>8B</td>
<td>13A</td>
<td>20B</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td>3A</td>
<td>9A</td>
<td>13B</td>
<td>21A</td>
</tr>
<tr>
<td>F6</td>
<td>3B</td>
<td>9B</td>
<td>14A</td>
<td>21B</td>
</tr>
<tr>
<td>F7</td>
<td>4A</td>
<td>10A</td>
<td>14B</td>
<td>22A</td>
</tr>
<tr>
<td>F8</td>
<td>4B</td>
<td>10B</td>
<td>15A</td>
<td>22B</td>
</tr>
<tr>
<td>Desired</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F9</td>
<td>5A</td>
<td>11A</td>
<td>16B</td>
<td>23A</td>
</tr>
<tr>
<td>F10</td>
<td>5B</td>
<td>15B</td>
<td>17A</td>
<td>23B</td>
</tr>
<tr>
<td>F11</td>
<td>6A</td>
<td>16A</td>
<td>17B</td>
<td>24A</td>
</tr>
<tr>
<td>F12</td>
<td>6B</td>
<td>18A</td>
<td>18B</td>
<td>24B</td>
</tr>
</tbody>
</table>

Team Utilization

- Make sure to use the skills of each team member well
  - All eggs in one basket
  - Jack of all traits, master of none
- Keep everyone busy
  - Now waiting, if possible
- Communication is vital
  - Every programmer should be aware of what others are doing
    - Code reviews
    - Joint status meetings
    - Documentation
Scheduling: Eggs in one Basket

<table>
<thead>
<tr>
<th>Feature</th>
<th>Null</th>
<th>Base</th>
<th>Nominal</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>1A</td>
<td>7A</td>
<td>12A</td>
<td>19A</td>
</tr>
<tr>
<td>F2</td>
<td>1B</td>
<td>7B</td>
<td>11B</td>
<td>19B</td>
</tr>
<tr>
<td>F3</td>
<td>2A</td>
<td>8A</td>
<td>13A</td>
<td>20A</td>
</tr>
<tr>
<td>F4</td>
<td>2B</td>
<td>8B</td>
<td>12B</td>
<td>20B</td>
</tr>
<tr>
<td>Required</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td>3A</td>
<td>9A</td>
<td>14A</td>
<td>21A</td>
</tr>
<tr>
<td>F6</td>
<td>3B</td>
<td>9B</td>
<td>13B</td>
<td>21B</td>
</tr>
<tr>
<td>F7</td>
<td>4A</td>
<td>10A</td>
<td>15A</td>
<td>22A</td>
</tr>
<tr>
<td>F8</td>
<td>4B</td>
<td>10B</td>
<td>14B</td>
<td>22B</td>
</tr>
<tr>
<td>Desired</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F9</td>
<td>5A</td>
<td>11A</td>
<td>16A</td>
<td>23A</td>
</tr>
<tr>
<td>F10</td>
<td>5B</td>
<td>15B</td>
<td>16B</td>
<td>23B</td>
</tr>
<tr>
<td>F11</td>
<td>6A</td>
<td>17A</td>
<td>18A</td>
<td>24A</td>
</tr>
<tr>
<td>F12</td>
<td>6B</td>
<td>17B</td>
<td>18B</td>
<td>24B</td>
</tr>
</tbody>
</table>

Scheduling with Iteration

- **Shift:**
  - FROM: When will it be finished?
  - TO: When will it be good enough?
- "Finished" is meaningless anyway
- We have a definition of "Good Enough" now!
- Bad estimation often comes from top-down dissection
  - No accounting for the learning curve, code revision, or integration
- **Iterative development**
  - Total time equals the sum of the Null, Base, Nominal, and Optimal levels