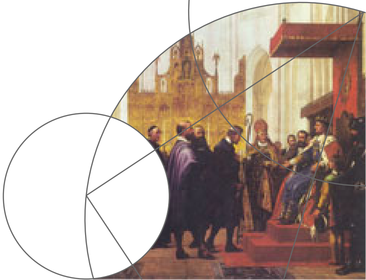
Faculty of Science

Space Taxi: Physics

Software Development

Anders Geil (xkh299)



1. Objectives

1. Physics-based movement.
2. Swift key response.
3. Simultaneous key presses.
4. Asynchronous key presses and releases.
5. Animations matching movements.
6. Potential support for multiple users.
7. Maintainable, robust API.

2. Design

Three core dilemmas:

1. High key input complexity, yet low response time.
2. Quick animation shifting, yet (potentially) heavy physics computation.
3. Robustness and maintainability, yet flexibility and extensibility.

2. Design

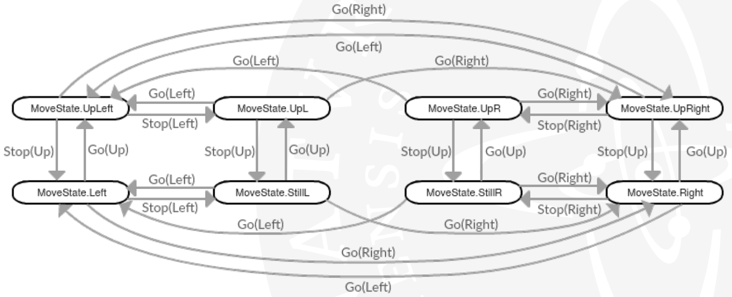
Three core dilemmas:

1. High key input complexity, yet low response time.
2. Quick animation shifting, yet (potentially) heavy physics computation.
3. Robustness and maintainability, yet flexibility and extensibility.

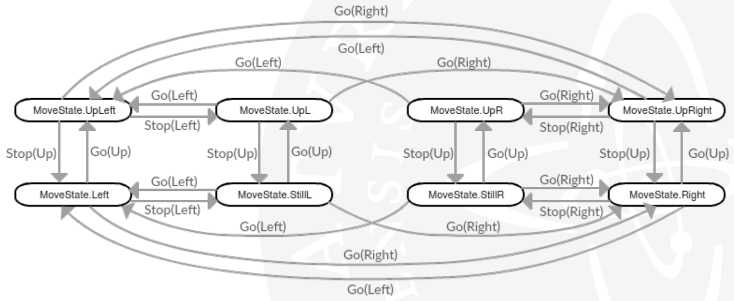
Three core solutions:

1. Emulation of keyboard states; utilize flagged enums and bitmasking.
2. Separation of concerns; handle graphics and physical computations independently.
3. Draw on accumulated experience; establish strategy design pattern.

3. Implementation (solution 1)



3. Implementation (solution 1)



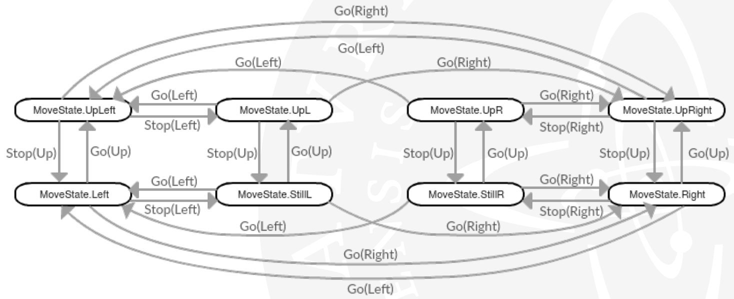
Ex: From *StillL* to *Right* using bitmasking

$Left = 1000$, $Still = 0100$, $Up = 0010$, $Right = 0001$

$Go(Right) = MoveState \vee Right \wedge \neg Left \wedge \neg Still$

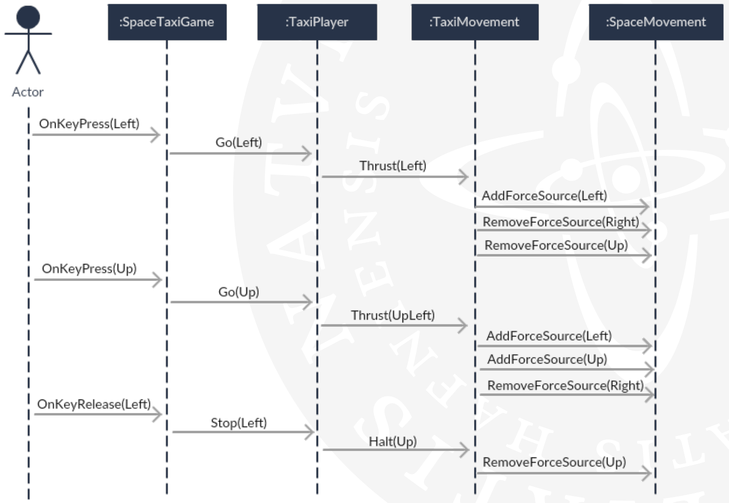
$Go(Right) = 1100 \vee 0001 \wedge 0111 \wedge 1011 = 0001$

3. Implementation (solution 1)

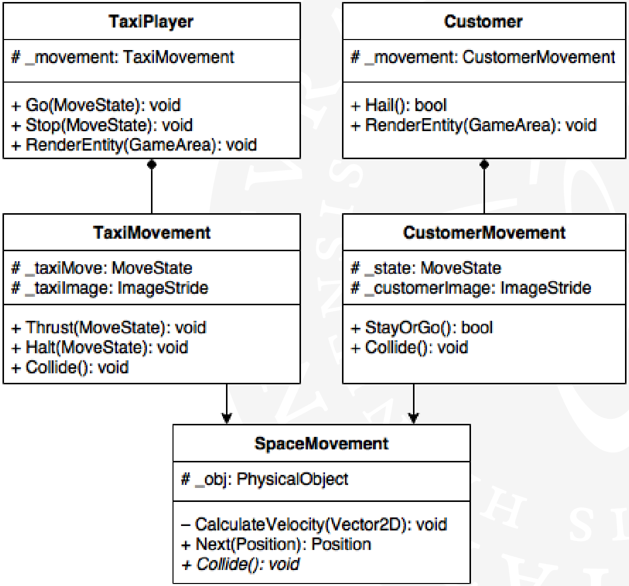


- Expresses all states as combinations of four states.
- Operations are independent of previous states.
- Reduces code complexity to two methods.

3. Implementation (solution 2)



3. Implementation (solution 3)



4. Evaluation

1. Unit testing: Core units/components
   - Ex: Methods changing MoveState or animations.
2. Integration testing: Simple unit interactions
   - Ex: Changing MoveState also changes animation.
3. System testing: Complex unit/component interactions
   - Ex: Taxi landing/crashing on platforms/customers.
4. Acceptance testing: Formal ’customer’ requirements
   - Ex: Mathematical specifications for movement behaviour within 3 frames.

5. Concluding Remarks

1. Define abstract move methods in space movement.
2. Refactor surrounding methods into space movement component.
   - Integrate customers' block collision detection into physics module (similarly to the taxi).
   - Load portals like solid surfaces in the abstract movement class (added benefit: customers could drop into portals too!).
3. Utilize strategy pattern better externally (e.g. not update player and customer positions separately).
4. Expand customer movement strategies (e.g. different customer psychologies – not all are suicidal!).
5. Implement further gameplay features (e.g. pickups).