Lecture 4a

Vectors and Steering Behaviour

Alan Dorin

FIT3094 Artificial Life, Artificial Intelligence and Virtual Environments

MONASH University
Information Technology

Learning Objectives

To know how to apply steering behaviour to game AI.

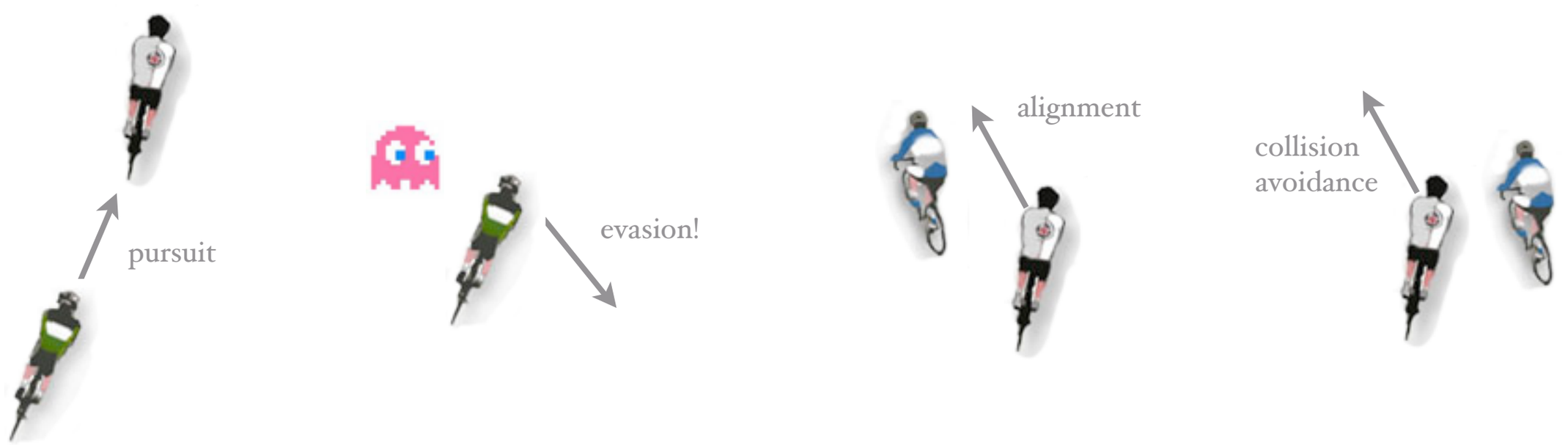
To understand the application of bounding boxes in collision detection.

To know how to compute vectors for agent pursuit, evasion and collision avoidance.

Steering Behaviour

Steering behaviour is an individual agent's response to stimuli by movement.

Turning when? Which direction? How much?
Moving when? How far? How fast?

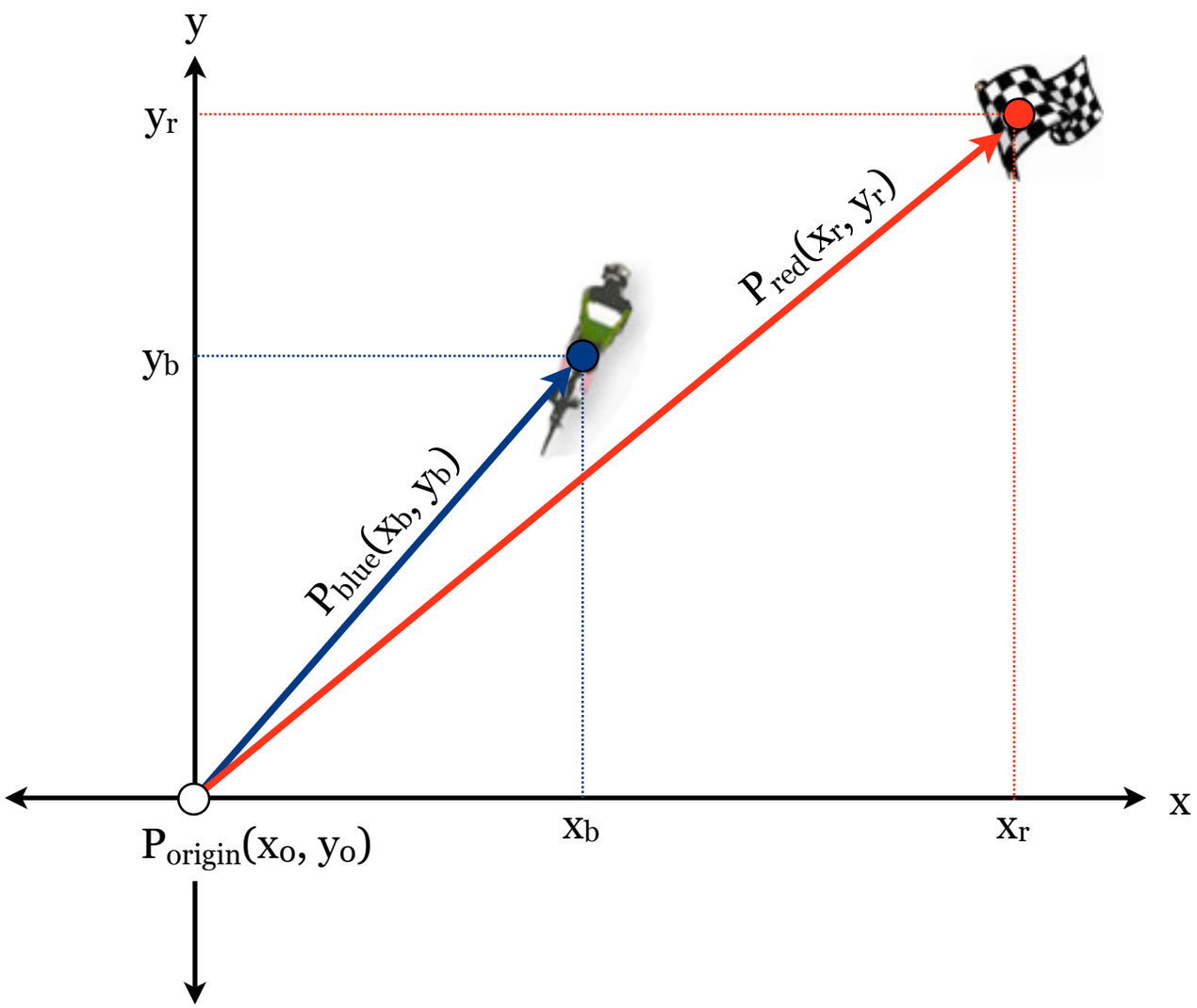


To give an agent the appearance of intelligence, it will need to steer in a way that is appropriate to its current knowledge, decision-making and physical capabilities.

Steering behaviour : vectors

To compute steering behaviour we usually use *vector* arithmetic.

vector
noun
Mathematics & Physics a quantity having direction as well as magnitude, esp. as determining the position of one point in space relative to another.



A vector in 2D space has x and y *components*.

The *magnitude* of a vector corresponds to its length — the distance between its endpoints:

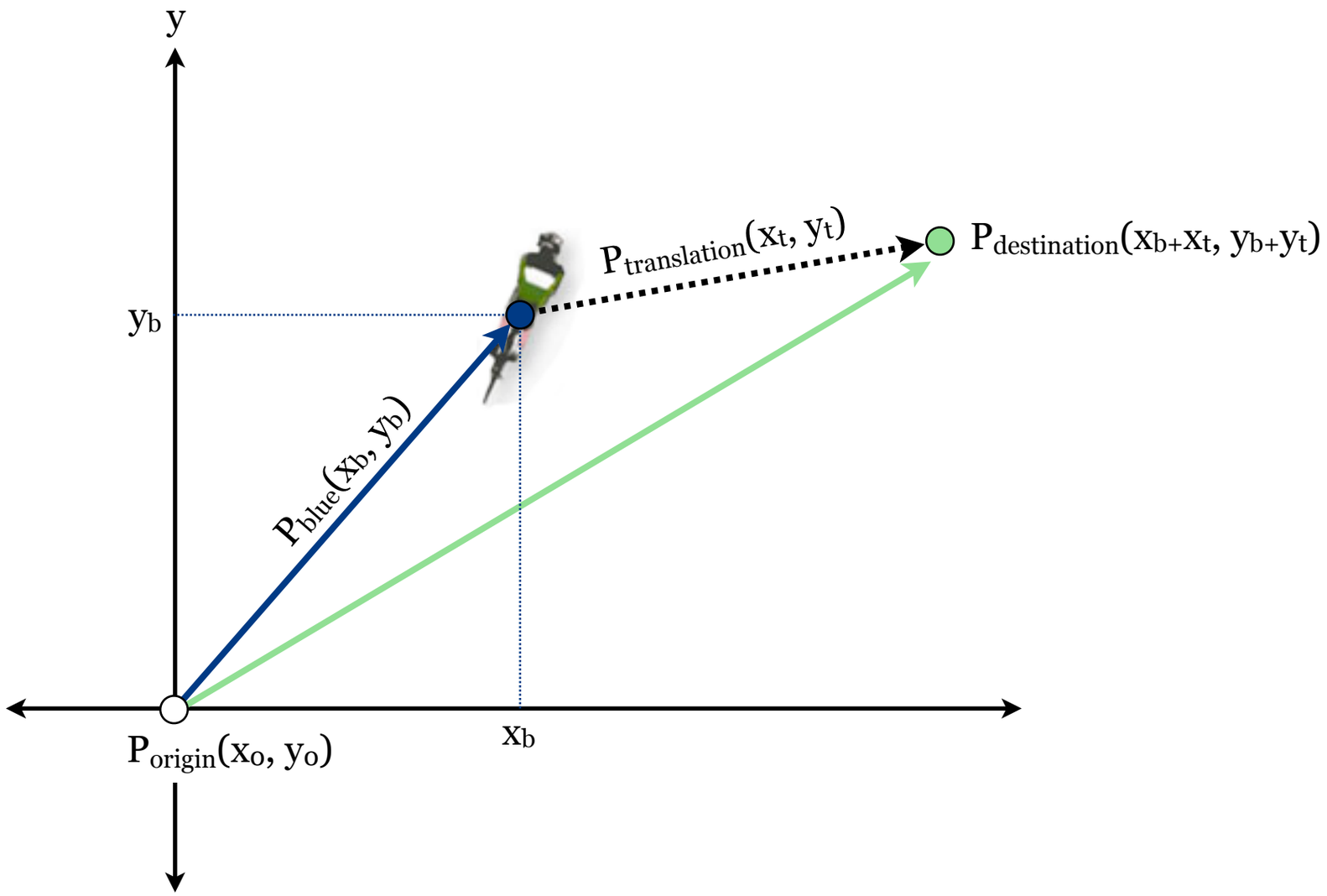
$$|\boldsymbol{P}| = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$$

What is the magnitude of P_{blue} in the diagram?

Steering behaviour : vector translation

To *translate* an object — i.e. to move the object — add a vector to the vector representing its current position.

If this translation represents the movement of an agent, we can say that this vector is the agent's *velocity* vector.

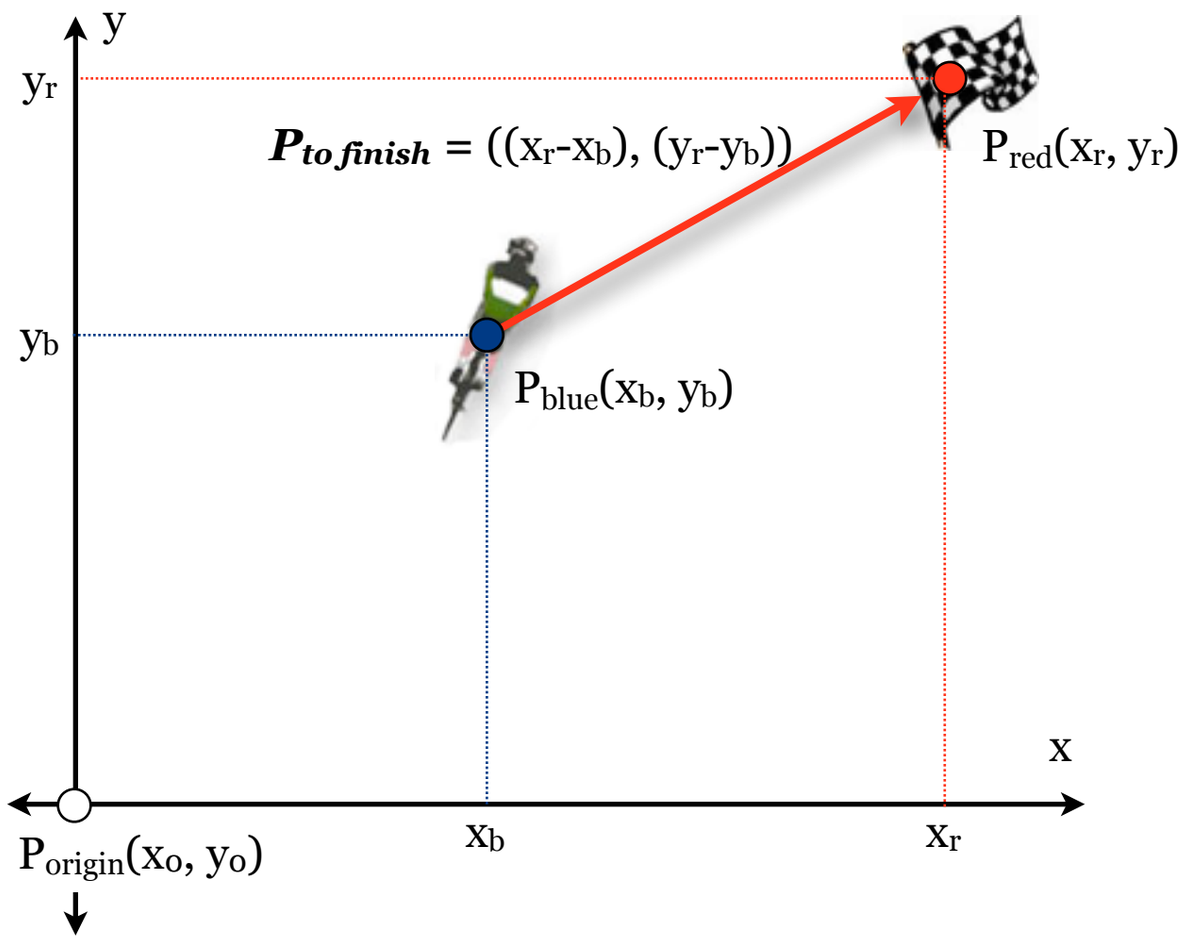


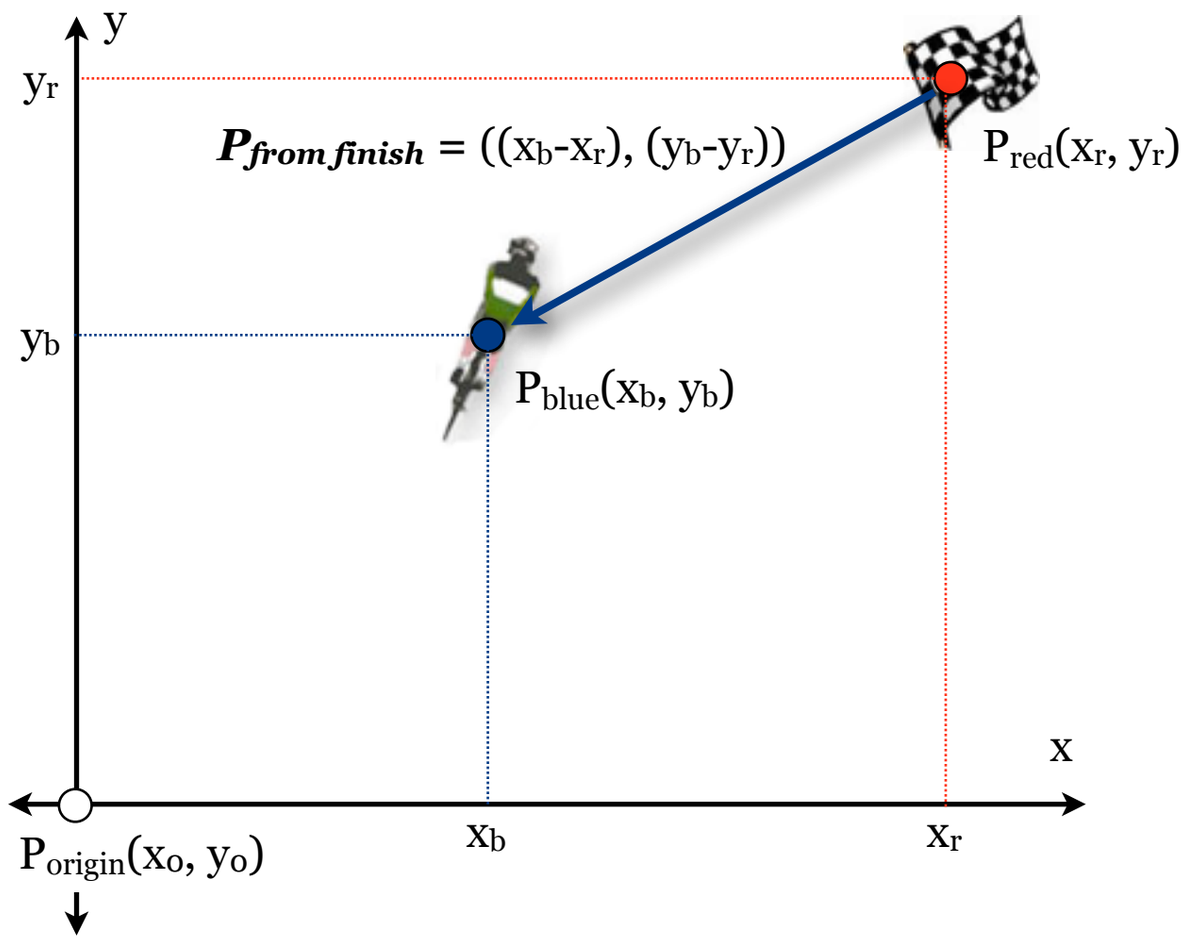
Steering behaviour : vectors

To compute a vector between two points remember the simple rule:
The vector *between* two points is the vector of the desired final position *minus* the vector of the desired initial position…

…or just remember the expression "*final minus initial*"

$$\boldsymbol{P_{from\ 1\ to\ 2}} = ((x_2\text{-}x_1), (y_2\text{-}y_1))$$





Steering behaviour : pursuit

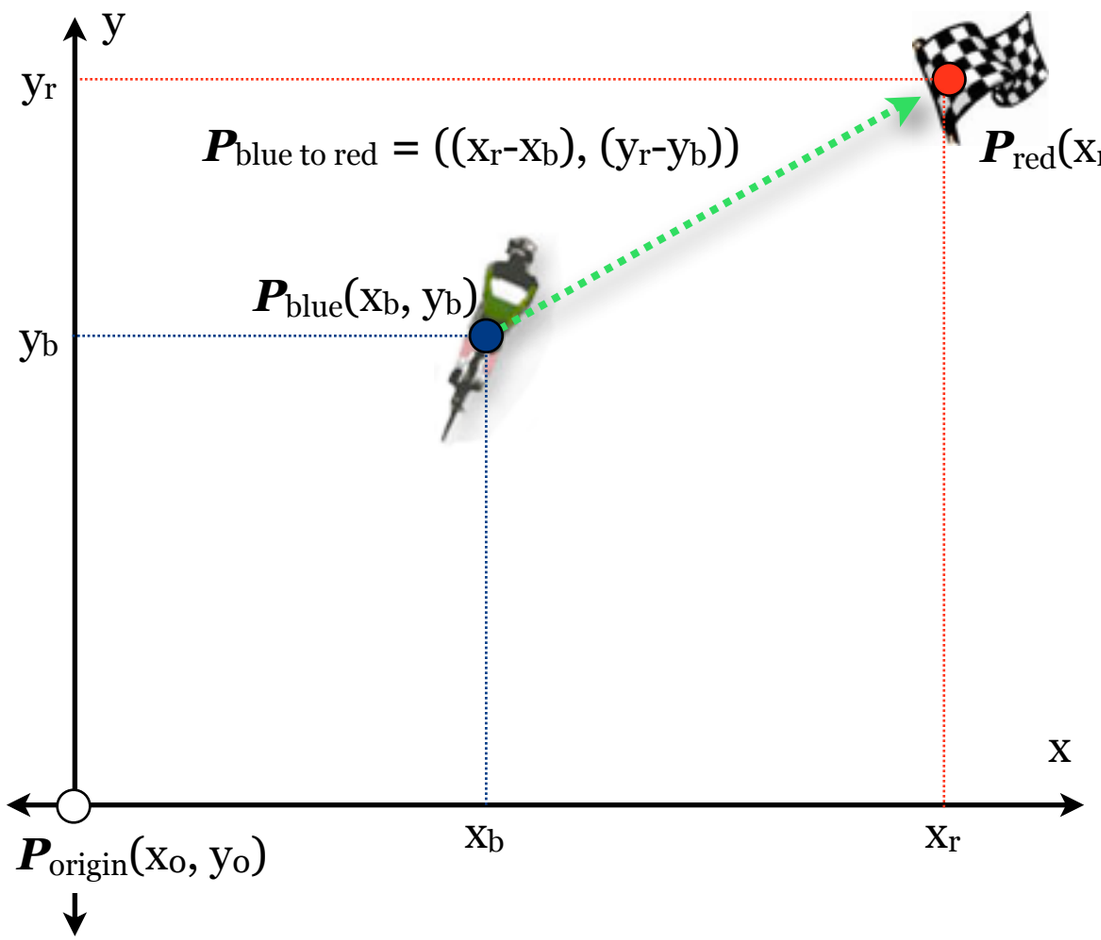
Pursuit requires the calculation of a vector *towards* a target and *translation* in that direction.

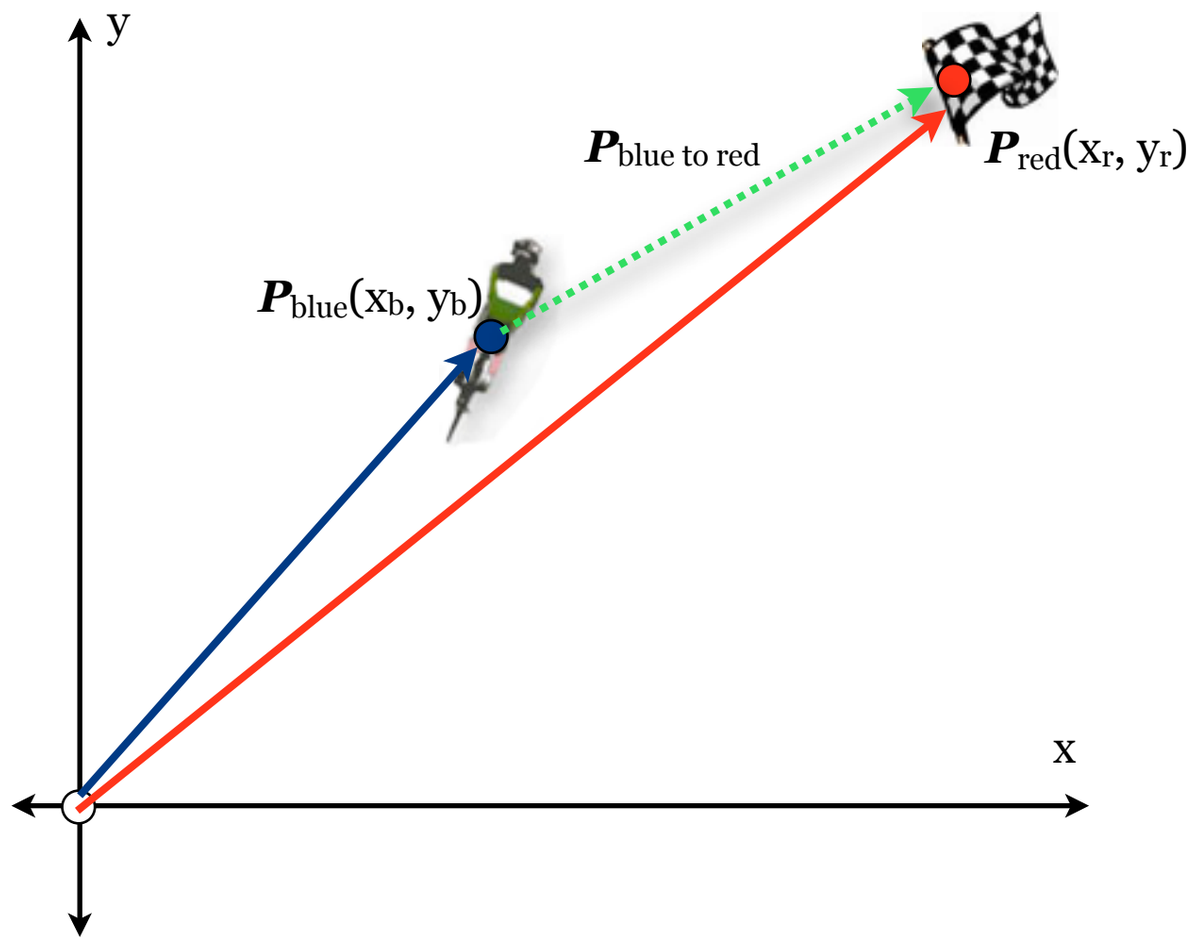
E.g., To move the blue cyclist to the finishing flag,

compute a vector *from* the cyclist *to* the flag...

...add the computed vector to the cyclist's position.

$$\boldsymbol{P}_{\text{blue destination}} = \boldsymbol{P}_{\text{blue}} + \boldsymbol{P}_{\text{blue to red}}$$

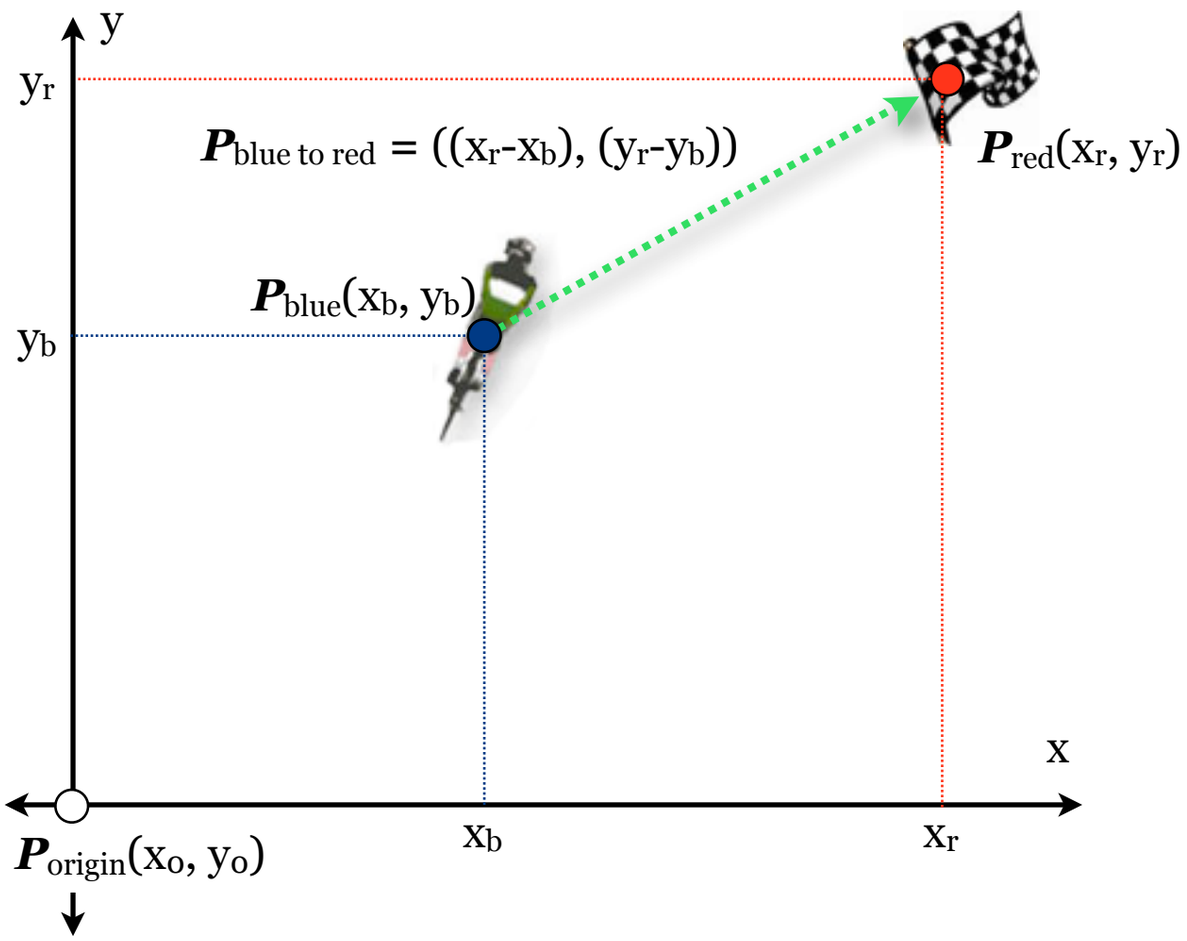




Steering behaviour : pursuit

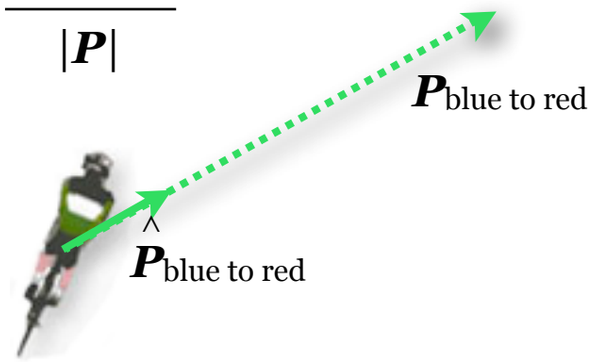
It is unrealistic for the blue cyclist to instantly move to the finish! We should only move the blue cyclist towards the flag by an amount governed by the blue cyclist's speed.

As before, we must still compute a vector *from* the blue cyclist *to* the flag.

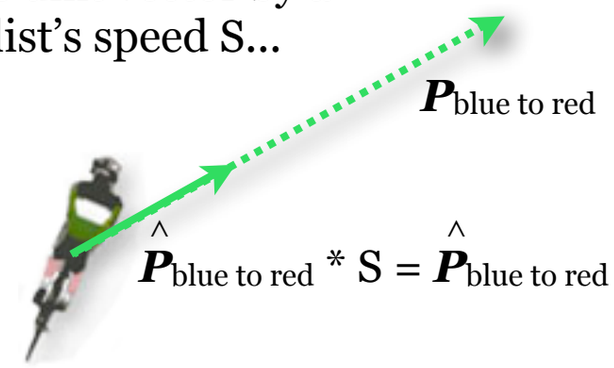


But we should then derive from this a *unit vector* (a vector of *magnitude* 1) in the direction of this vector...

$$\hat{\boldsymbol{P}}_{\text{blue to red}} = \frac{\boldsymbol{P}_{\text{blue to red}}}{|\boldsymbol{P}|}$$



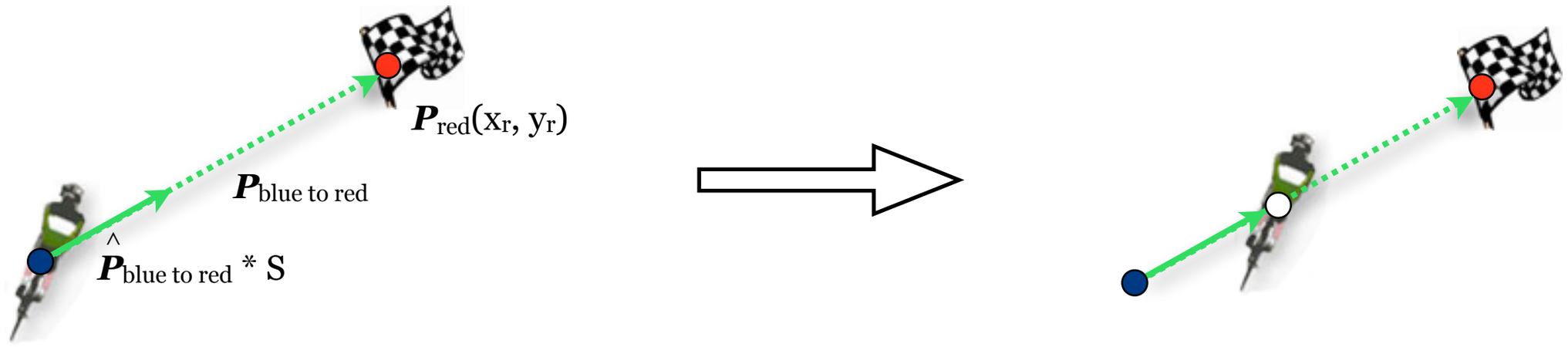
We multiply the unit vector by a scalar*, the cyclist's speed S...



...and add *this* velocity vector to the cyclist's position...

* A *scalar* is a number like "2.3" or "77". It has a magnitude, but *no* direction.

Steering behaviour : pursuit



before translation by velocity

after translation by velocity

new position = ***old position*** + (***velocity*** * time-step)

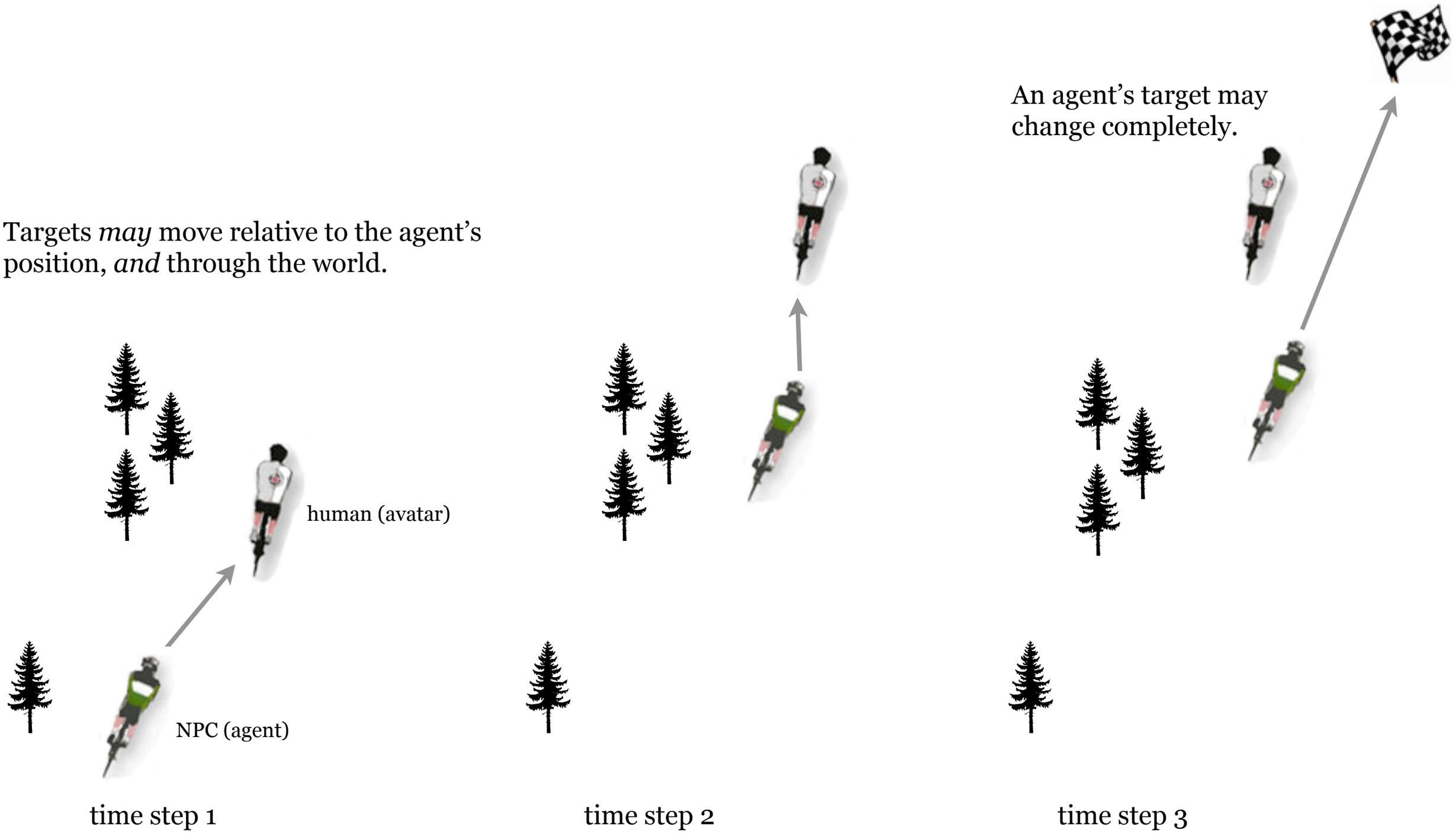
Euler integration says: $\boldsymbol{u}' = \boldsymbol{u} + \boldsymbol{v}t$

Steering behaviour : pursuit

After every update of a game world, each pursuing agent needs to compute a new velocity vector based on its speed and the direction to its target.

Targets *may* move relative to the agent's position, *and* through the world.

An agent's target may change completely.



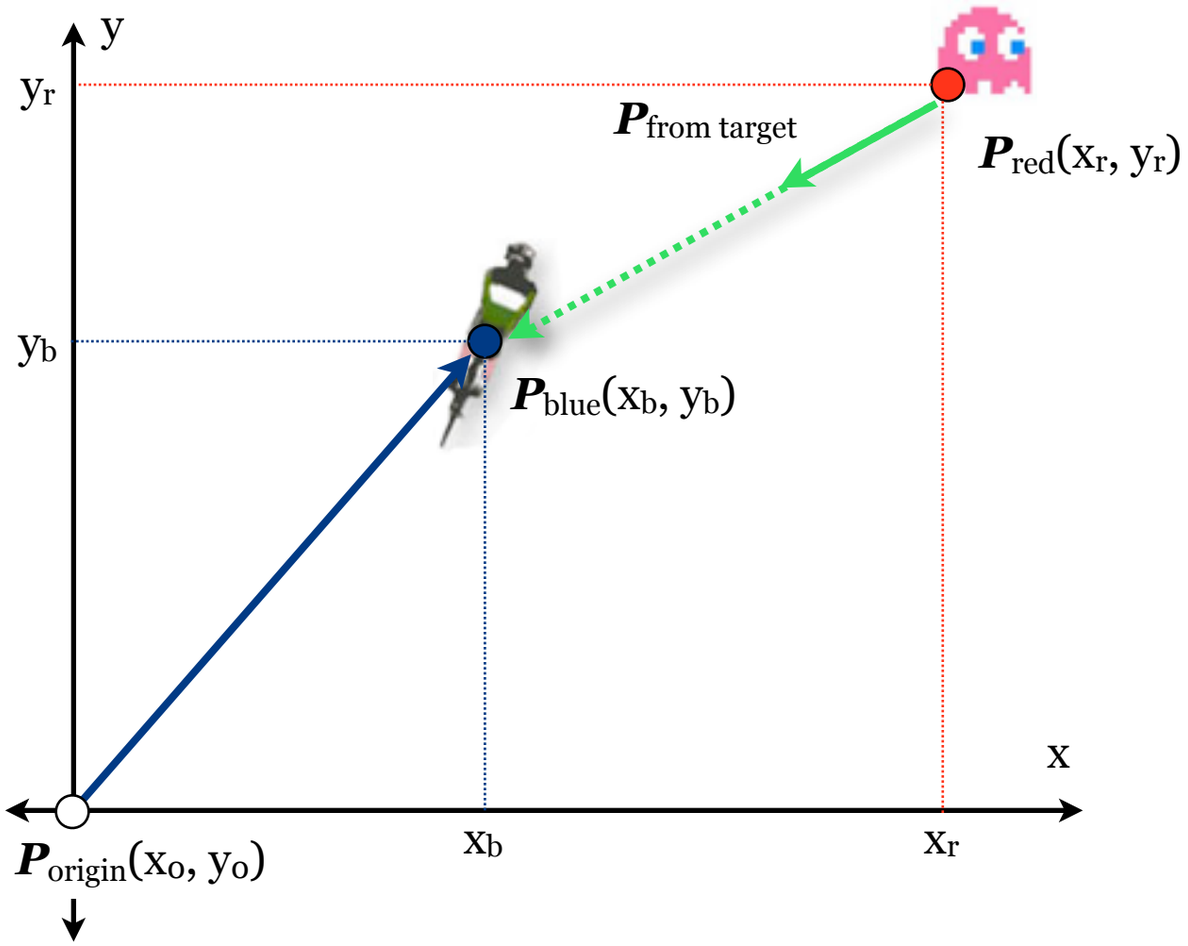
Steering behaviour : evasion

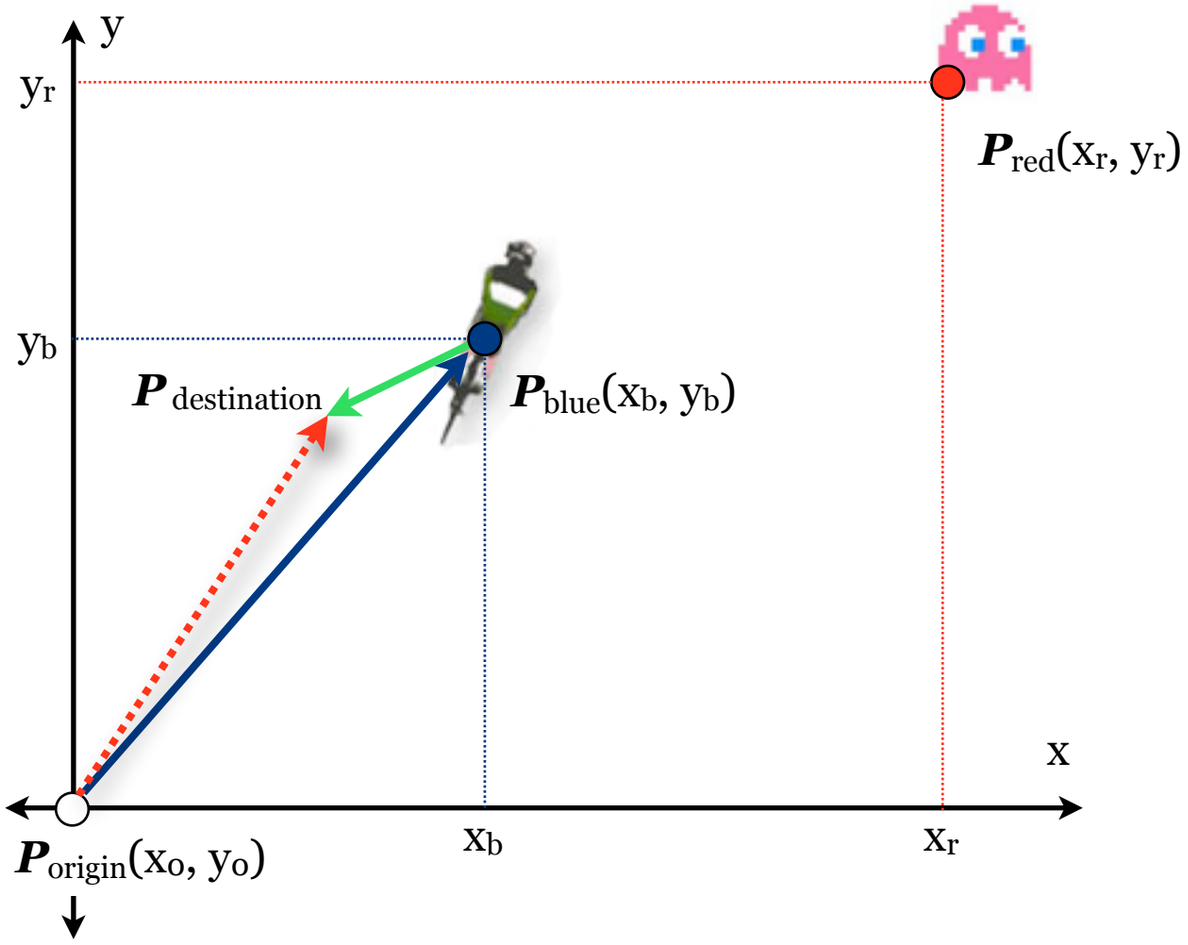
Evasion requires the calculation of a velocity vector *away from* a point.

E.g., To move the blue cyclist away from the ghost,

compute a unit vector *from* the ghost *to* the cyclist and scale it by the cyclist's speed...

...as before, add the computed velocity vector to the cyclist's position*.

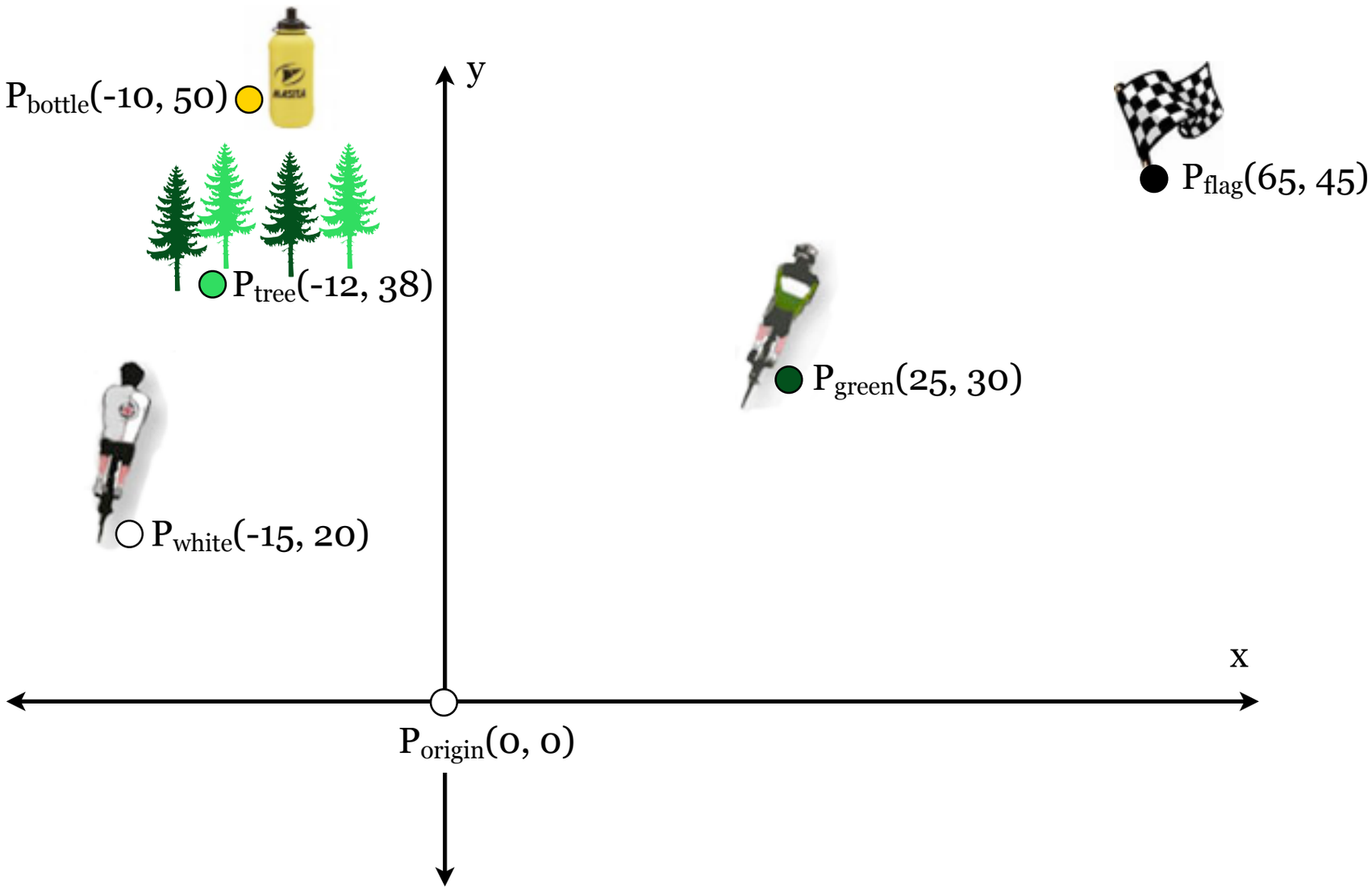




* Is it realistic for a cyclist to change direction like this?

Steering behaviour : exercise

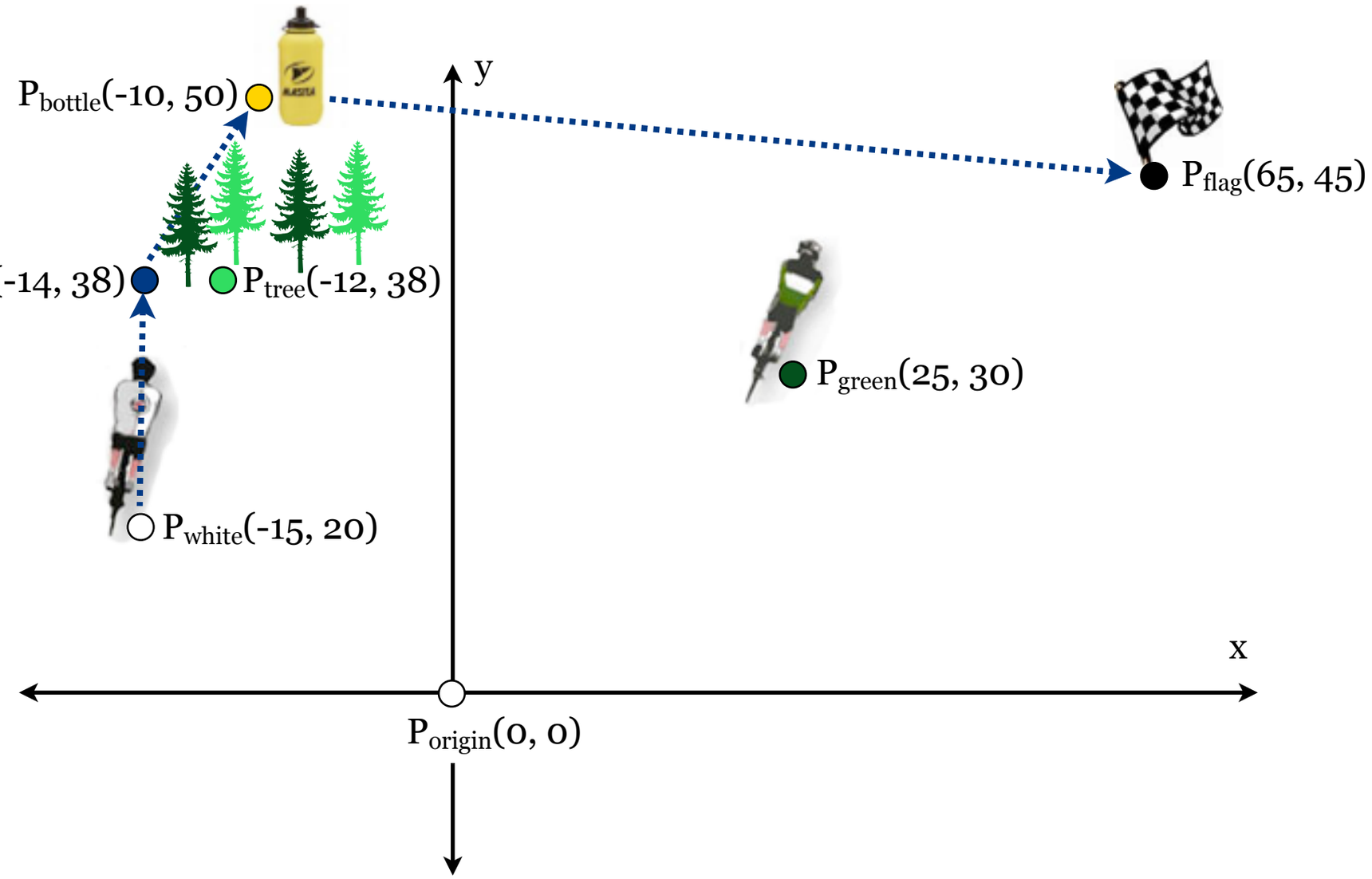
Compute a vector from the green cyclist to the finish line.
At a speed of 2 units per time step, how many steps will it take the green cyclist to reach the flag?



If the white cyclist has to go around the left of the trees (collision avoidance) and collect the bottle on his way to the flag, how far does he have to travel to finish?

Steering behaviour : exercise

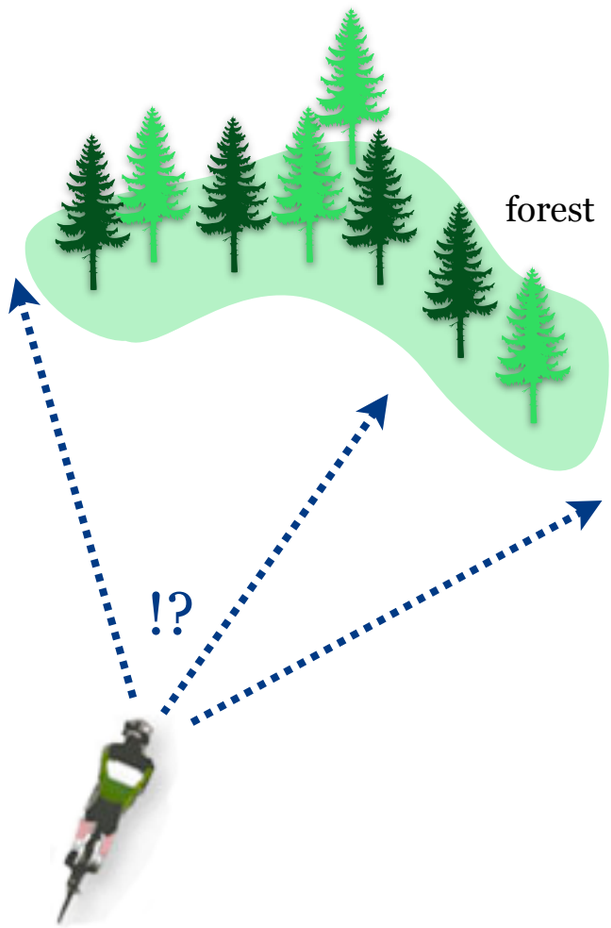
Tip: Create a *waypoint* to the left of the trees (allow space for the width of the cyclist and the trees).



Use this as an intermediate *goal* or target position before moving on to the next goal (the waypoint at the bottle) and then the final goal, the flag.

Steering behaviour : collision detection

Suppose an agent needs to avoid a collision with a wide object.



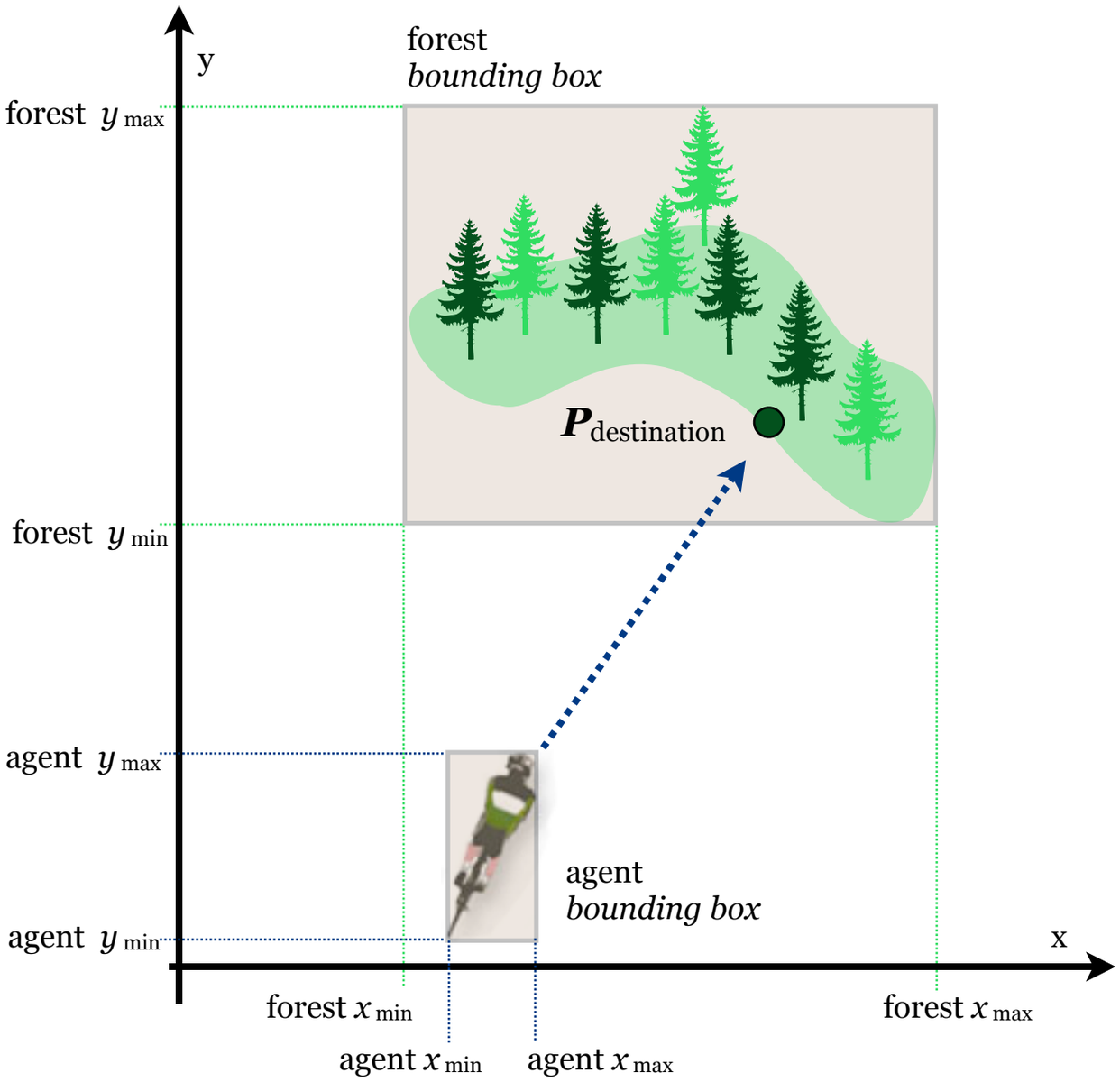
How does the agent *know* it needs to avoid a collision?

How does the agent *choose* which direction to go to avoid a collision?

Q: How does an agent *know* it needs to avoid a collision?

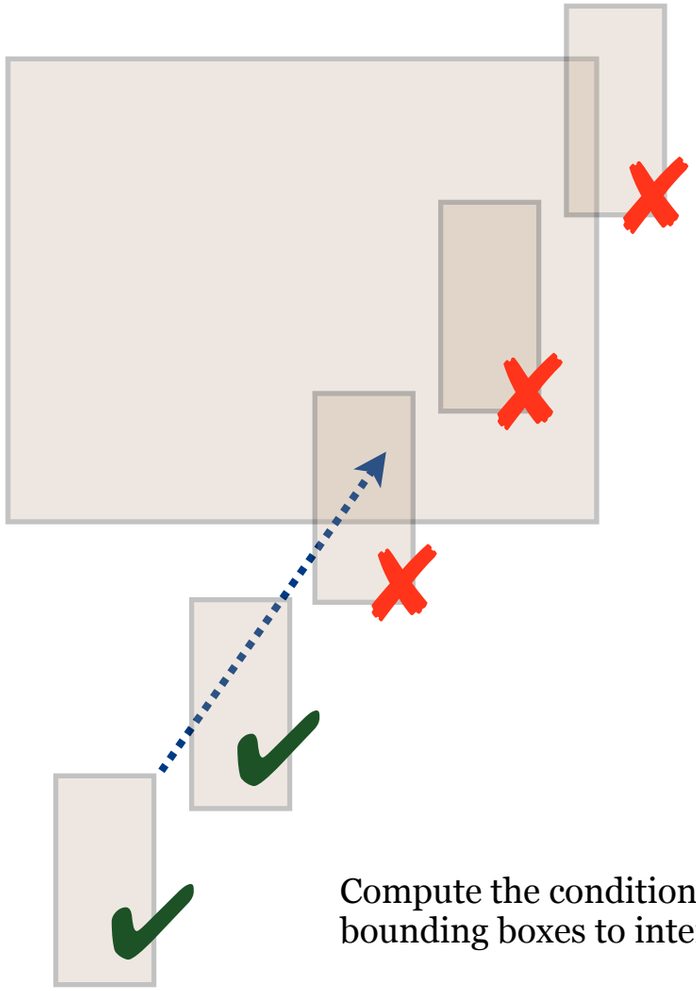
A: Its anticipated path *intersects* an obstacle.

Do the *bounding boxes* of the agent and obstacle intersect during the next simulation time step?



If the simulation time step is short, so that the agent moves only a small distance each time step, an *approximate* intersection test can be made...

Will the bounding boxes of the agent and obstacle intersect at the *end* of the time step?

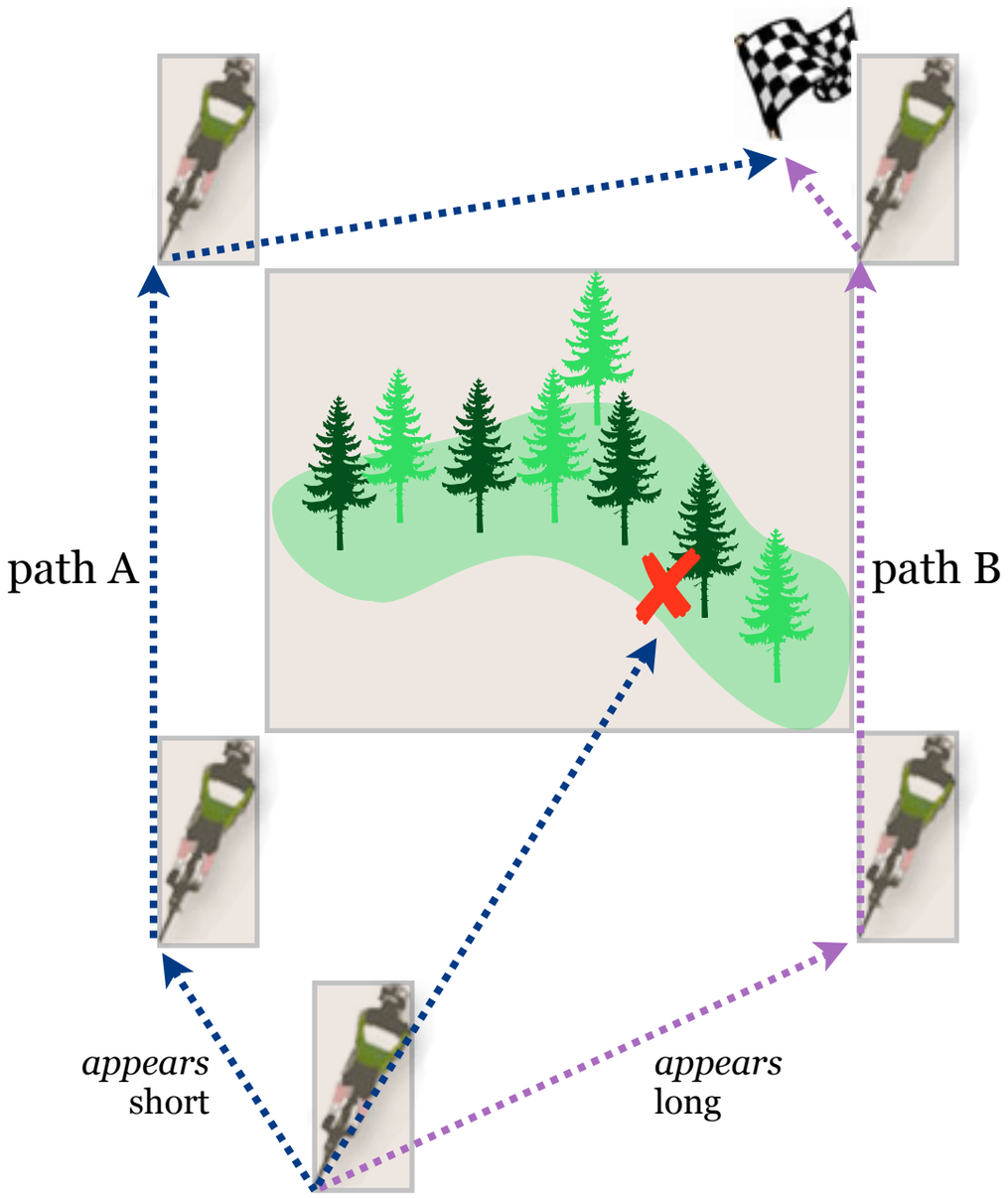


Compute the conditions for the bounding boxes to intersect!

What might happen if the time steps are too long?

Q: How does the agent *choose* which direction to go to avoid a collision?

A: It (probably) chooses the path that causes it to deviate the least.



Which path is this?!

The answer depends on:

- What the agent knows about the world
- The location of its final destination
- The agent's turning ability

This is a search problem*

A "simple" agent might choose path A.
A "smart" agent might calculate the lengths of all possible paths and prefer path B.

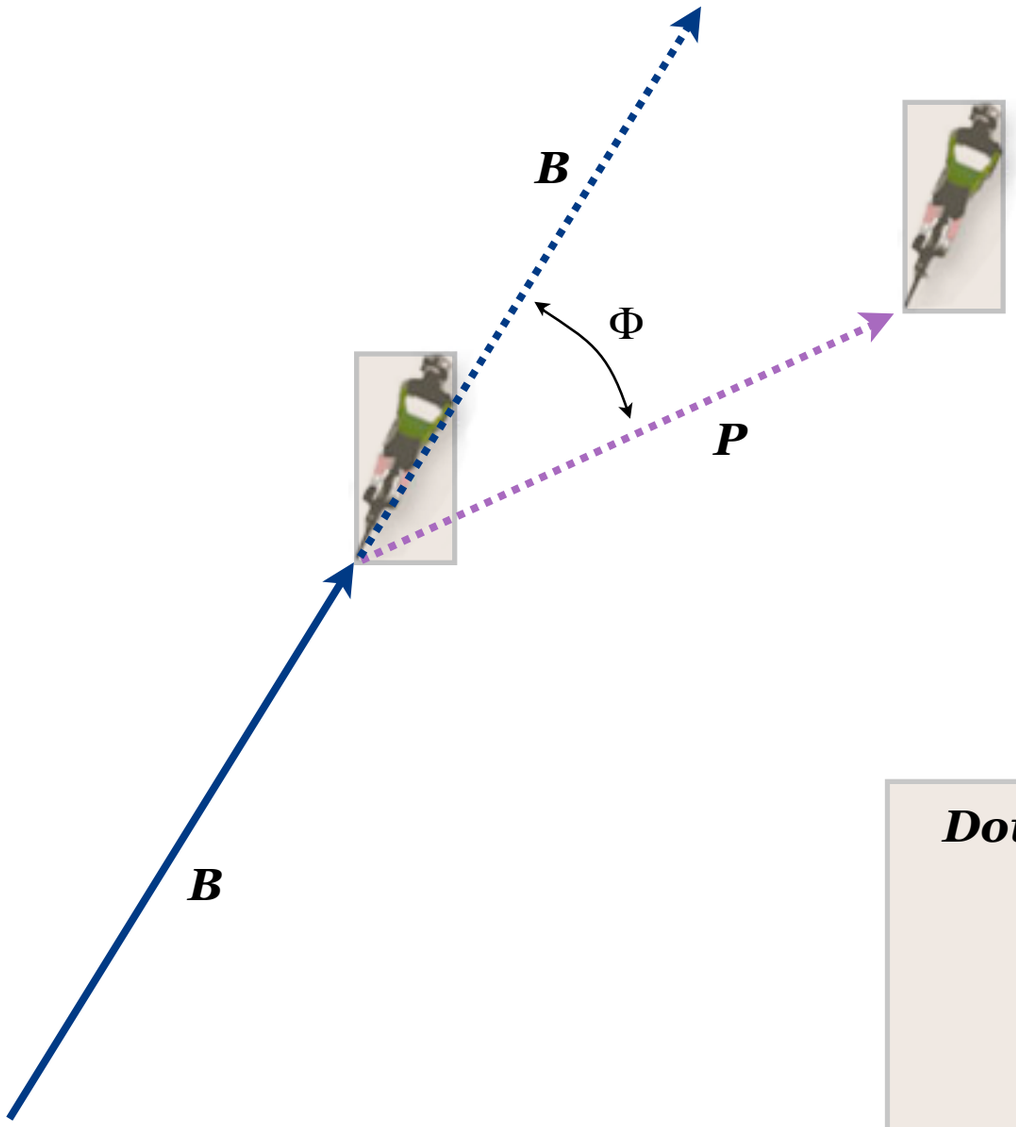
A fast agent might be forced to choose path B because it would be unable to turn sharply enough to take path A.

* We look at searching in other lectures.

Calculating the angle between two vectors using the dot product.

To avoid having an agent change direction too sharply, you need to know the deviation in *direction* between a velocity vector and the one that follows it.

You want to know the *angle* between two vectors!



Suppose a cyclist has just been translated by vector ***B*** and has a choice now of translating by ***B*** again, or turning with ***P***.

What change in direction is needed to turn onto ***P*** ?

$$\Phi = \cos^{-1}(\hat{\boldsymbol{B}} \bullet \hat{\boldsymbol{P}}) = \cos^{-1}\left(\frac{\boldsymbol{B} \bullet \boldsymbol{P}}{|\boldsymbol{B}|\,|\boldsymbol{P}|}\right)$$

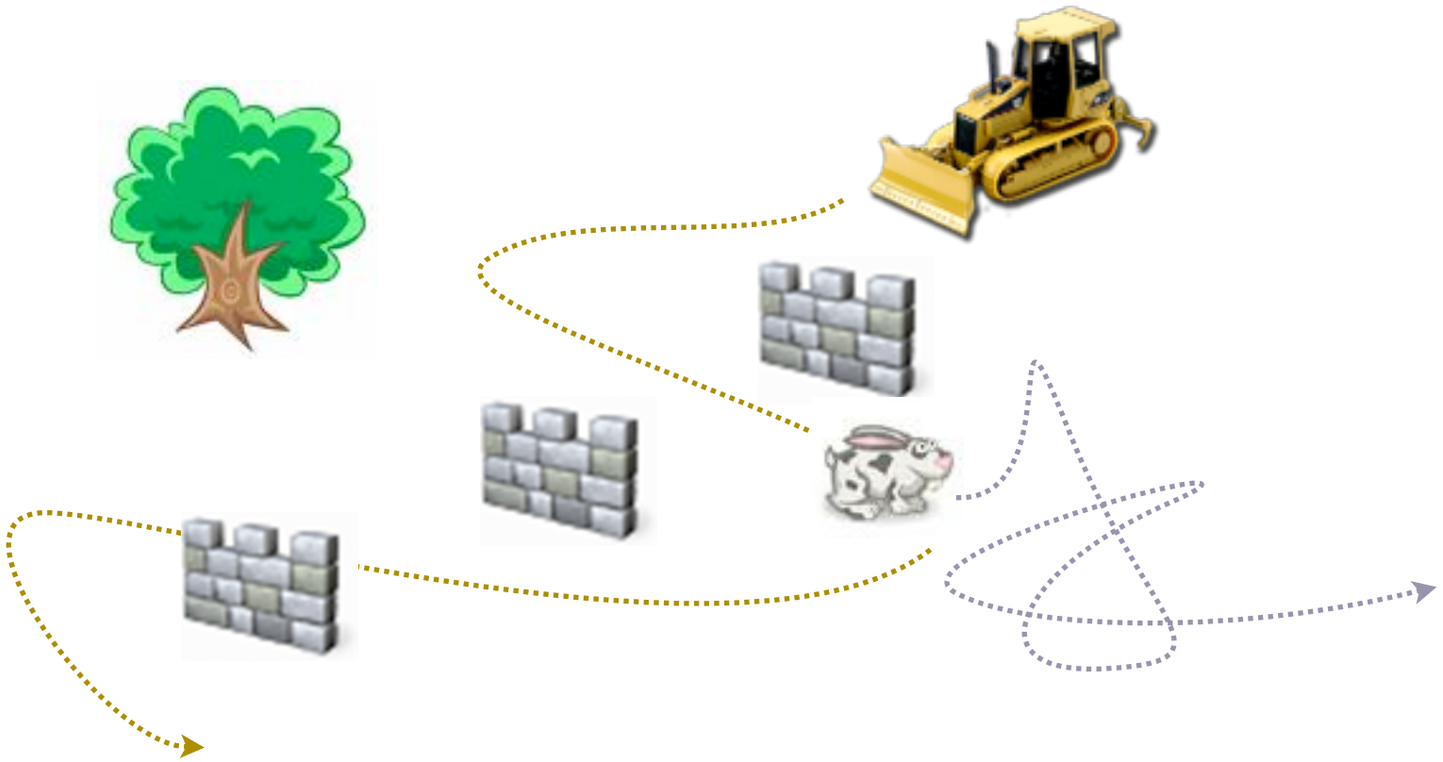
Dot Product

The "•" is the *dot product* of two vectors.
The result of this operation is a scalar value.

$$\boldsymbol{B} \bullet \boldsymbol{P} = |\boldsymbol{B}||\boldsymbol{P}|\cos(\Phi) = (x_b * x_p)+(y_b * y_p)$$

Steering behaviour

Tip: By carefully tweaking the angle of turn and frequency of directional changes, the speed and changes in speed of an agent, you can give the illusion of different mental states and physical properties.



Consider a cumbersome tractor driven by a bored farmer.
Consider a scared, agile rabbit.

How *often* do they change direction?
How *quickly* do they change direction?

Now consider a crazed tractor driver and a sleepy, dazed rabbit.
What changes?

Summary

Agents often need to make complicated movements around game worlds.
A way to compute these movements is with vector addition, subtraction and dot products.
A way to test for collisions between objects uses bounding boxes.

Can you explain the circumstances when each of these is required?
Can you compute some simple examples to show how these tools are used?