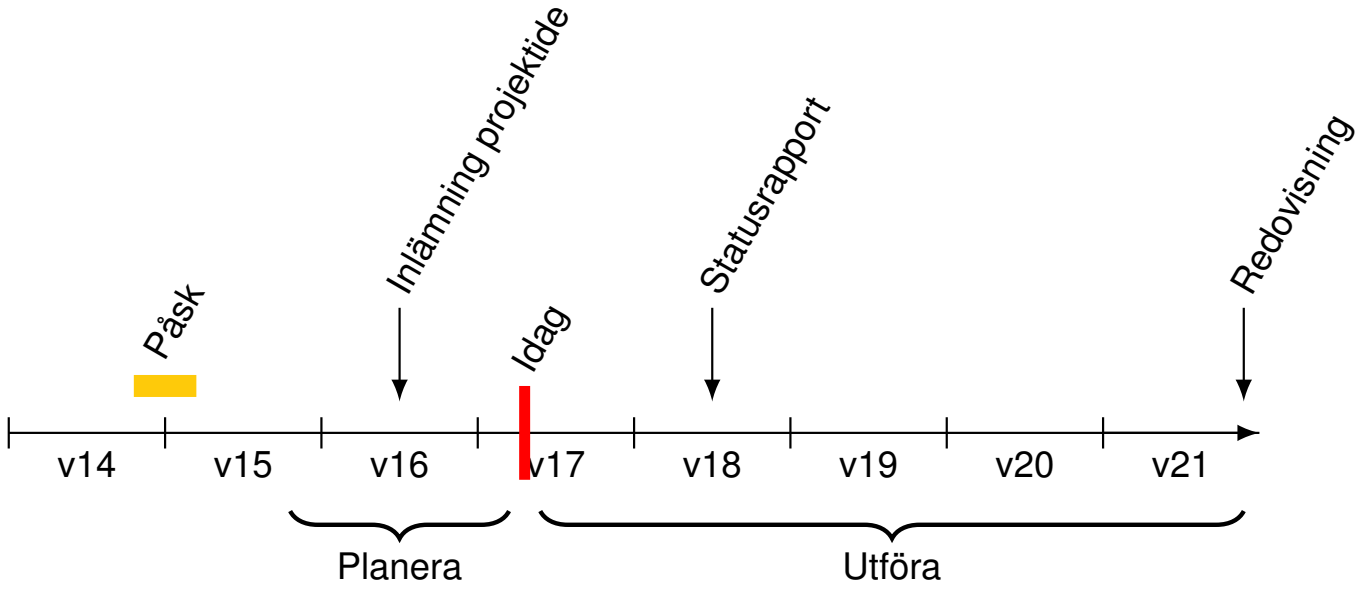
TDDI82
Spelutveckling och SFML

Eric Ekström

Institutionen för datavetenskap

Projektinformation - Tidslinje



Checklista

Innan idag:

- Lämna in gruppkontrakt och projektide
- Få projektide godkänd
- Sätta upp git och göra en commit

Efter idag:

- Göra klart malluppgiften
- Skapa ett UML-diagram
- Börja testa SFML
- Skapa en Makefil

1 Spelutveckling
2 Tillståndsmaskin
3 Resurshantering
4 Kollisionshantering
5 SFML
6 SFML Demo

LINKÖPING UNIVERSITY

Main-loop

```
int main()
{
  while (running)
  {
    handle_input();
    update_logic();
    render();

  }
}
```

Main-loop

```
int main()
{
  while (running)
  {
    handle_input();
    update_logic();
    render();
    throttle();
  }
}
```

1 Spelutveckling
2 Tillståndsmaskin
3 Resurshantering
4 Kollisionshantering
5 SFML
6 SFML Demo

Tillståndsmaskin (State machine)

Hur hanterar vi olika tillstånd i spelet?

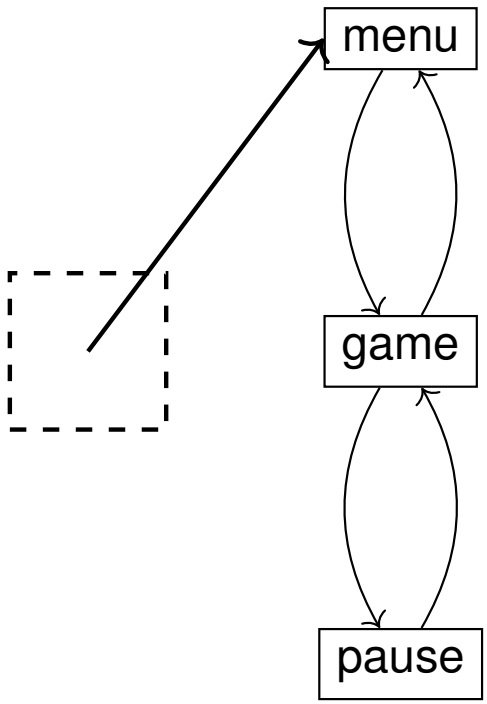
- Huvudmeny
- Inställningsmeny
- Faktiska spelet
- Gameover-skärm
- ...

Tillståndsmaskin (State machine)

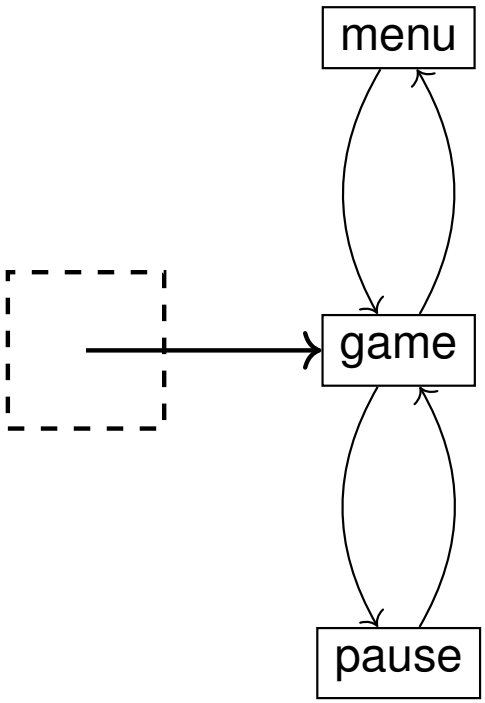
Naiv lösning:

```
int main()
{
  int state {};
  while (running)
  {
    if (state == 0)
      handle_menu();
    if (state == 1)
      handle_game();
    if (state == 2)
      handle_gameover();
  }
  throttle();
}
```

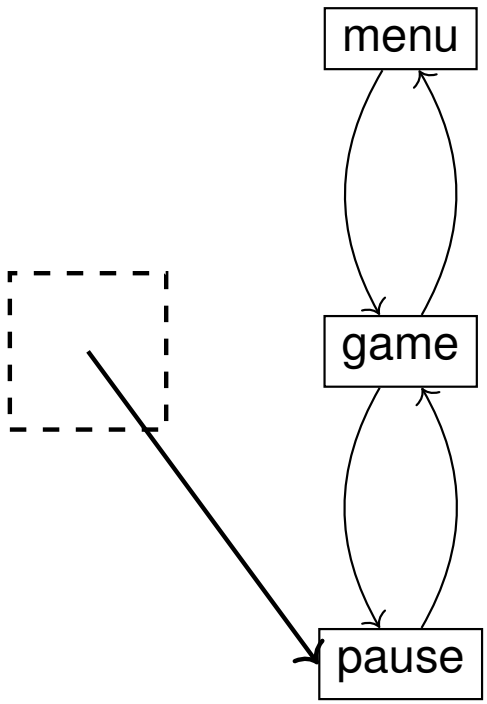
Tillståndsmaskin (State machine)



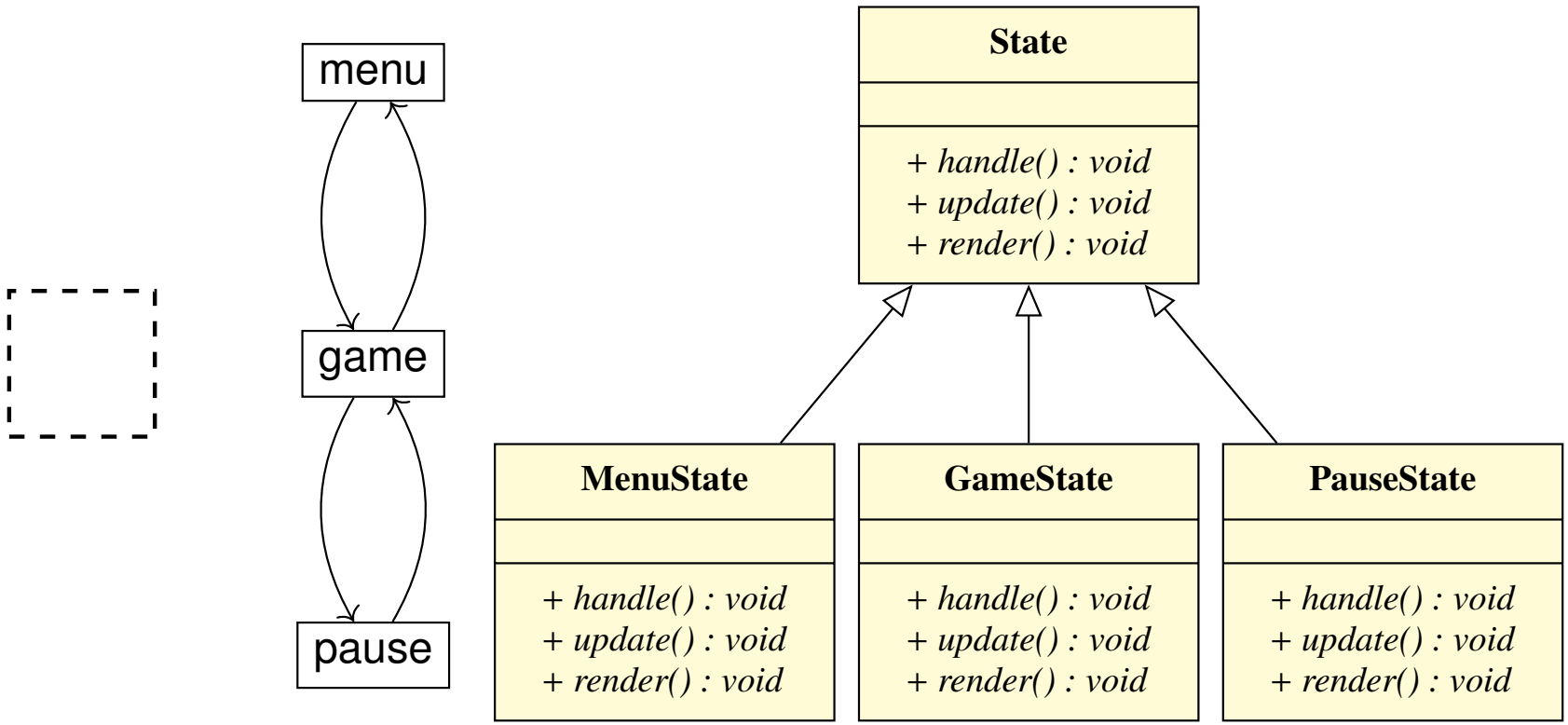
Tillståndsmaskin (State machine)



Tillståndsmaskin (State machine)



Tillståndsmaskin (State machine)



Tillståndsmaskin (State machine)

Objektorienterad lösning:

```
int main()
{
  State* state;
  while (running)
  {
    state->handle();
    state->update();
    state->render();
    throttle()
  }
}
```

Tillståndsmaskin (State machine)

- Det finns många sätt att göra en state machine
- Välj den som passar er bäst!
- **Tips:** Låt `update` returnera saker till tillståndsmaskinen
- Det finns exempel på kurshemsidan (Kopiera inte rakt av!)

1 Spelutveckling
2 Tillståndsmaskin
3 Resurshantering
4 Kollisionshantering
5 SFML
6 SFML Demo

LINKÖPING UNIVERSITY

Resurshantering

Vad är problemet här?

```
for (int i {}; i < 1000; ++i)
{
  enemies.push_back(Enemy{i*10, 0});
}

Enemy::Enemy(int x, int y)
  : x{ x }, y{ y },
    image{ load_from_file("enemy.png") }
{}
```

Resurshantering

Bättre lösning!

```
for (int i {}; i < 1000; ++i)
{
  enemies.push_back(Enemy{i*10, 0});
}

Enemy::Enemy(int x, int y)
  : x{ x }, y{ y },
    image{ Manager<Image>::load("enemy.png") }
{}
```

Resurshantering

```
template<typename T>
class Manager
{
  static map<string, T> resources;
public:
  static T& load(string const& file)
  {
    auto it { resources.find(file) };
    if (it == end(resources))
    {
      T res { load_from_file<T>(file) };
      it = resources.emplace(make_pair(file, res)).first;
    }
    return it->second;
  }
};
```

Resurshantering

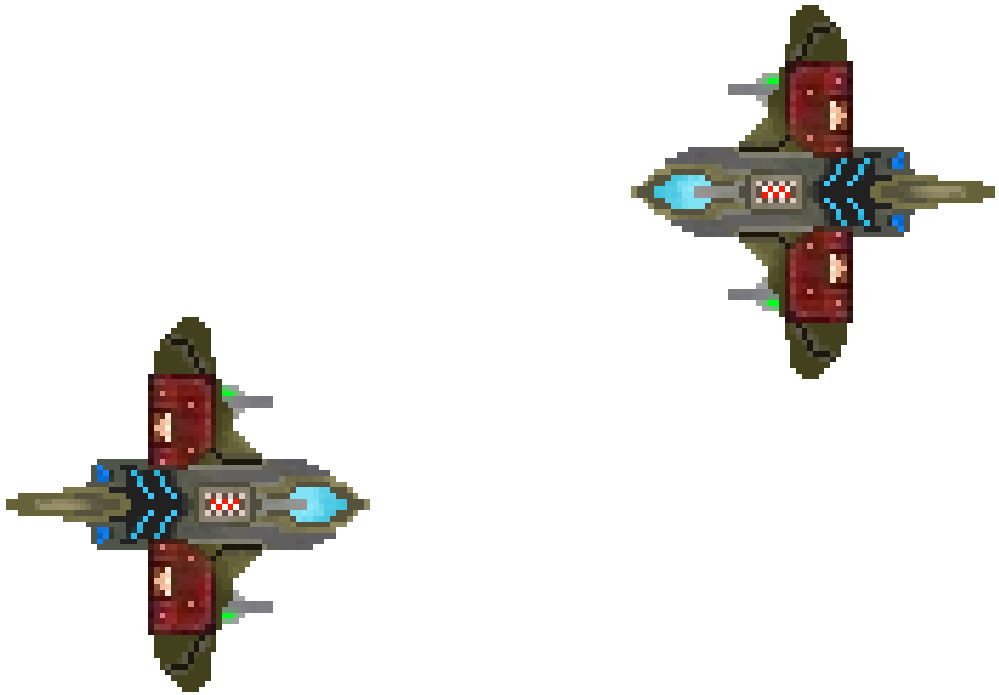
- Läs in varje resurs endast en gång
- Läs in resursen först när den behövs
- Spara referenser till vår `Manager`

1 Spelutveckling
2 Tillståndsmaskin
3 Resurshantering
4 Kollisionshantering
5 SFML
6 SFML Demo

LINKÖPING UNIVERSITY

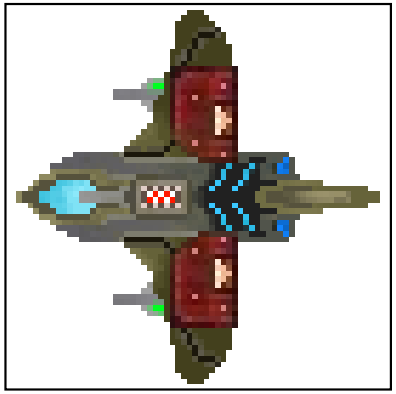
Kollisionshantering

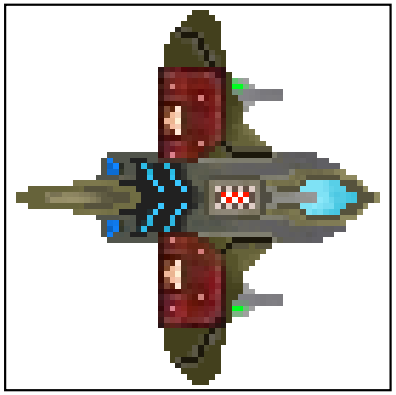
Problem: Hur vet vi om två objekt har kolliderat?



Kollisionshantering

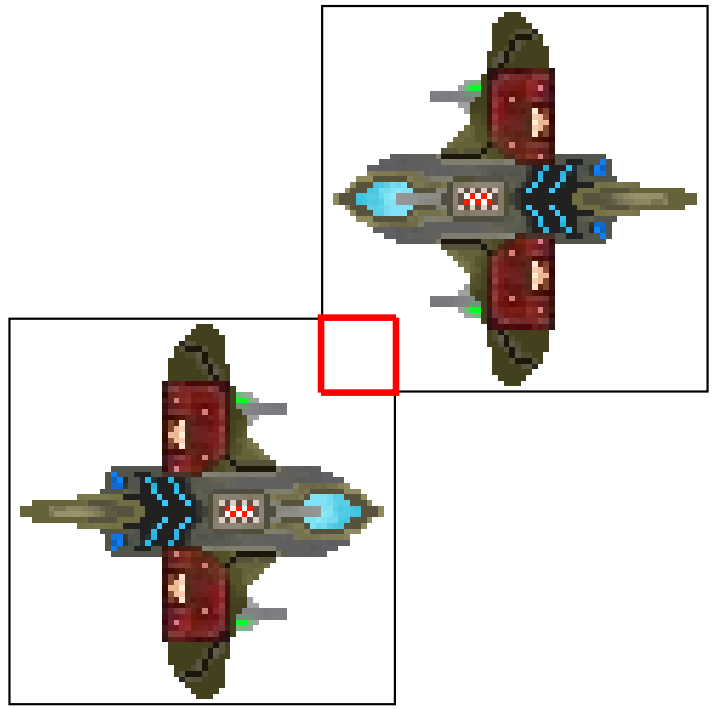
Lösning 1: Axis align bounding boxes (AABB)





Kollisionshantering

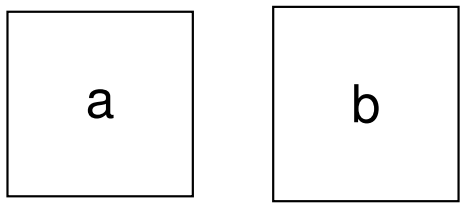
Lösning 1: Axis align bounding boxes (AABB)



Kollisionshantering

Lösning 1: Axis-Aligned Bounding Boxes (AABB)

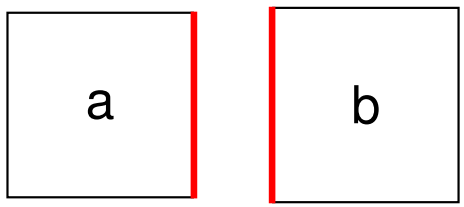
```
struct AABB
{
  int left;
  int right;
  int top;
  int bottom;
};

bool collide(AABB a, AABB b)
{
  return !(a.left > b.right ||
           a.top > b.bottom ||
           a.right < b.left   ||
           a.bottom < b.top);
}
```



Kollisionshantering

Lösning 1: Axis-Aligned Bounding Boxes (AABB)

```
struct AABB
{
  int left;
  int right;
  int top;
  int bottom;
};

bool collide(AABB a, AABB b)
{
  return !(a.left > b.right ||
           a.top > b.bottom ||
           a.right < b.left   ||
           a.bottom < b.top);
}
```

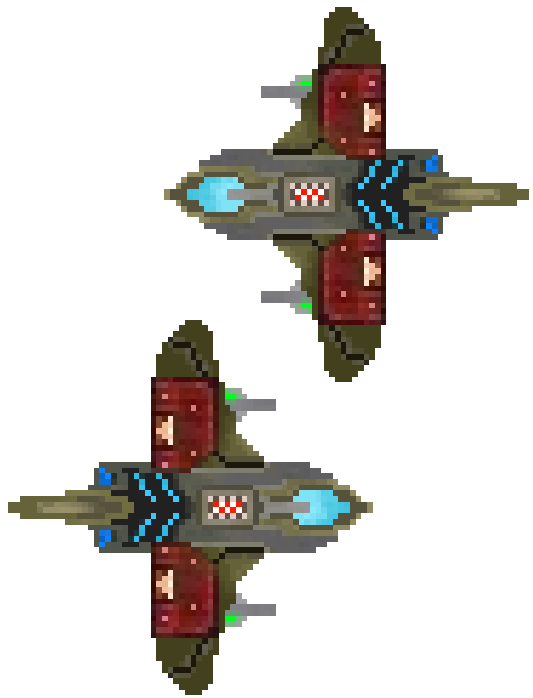


Kollisionshantering

Lösning 2: Pixel Perfect Collision

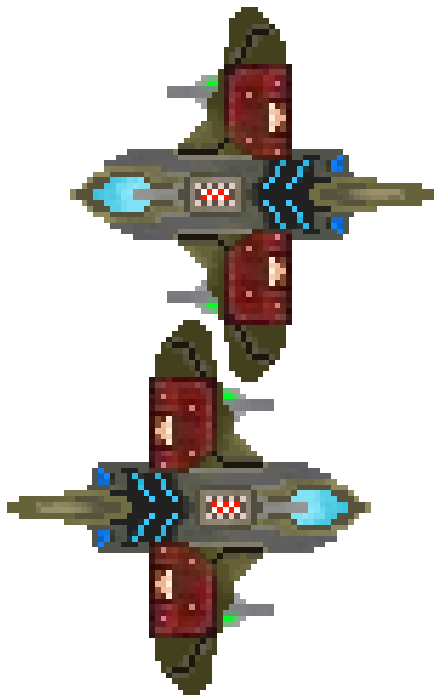
Kollisionshantering

Lösning 2: Pixel Perfect Collision



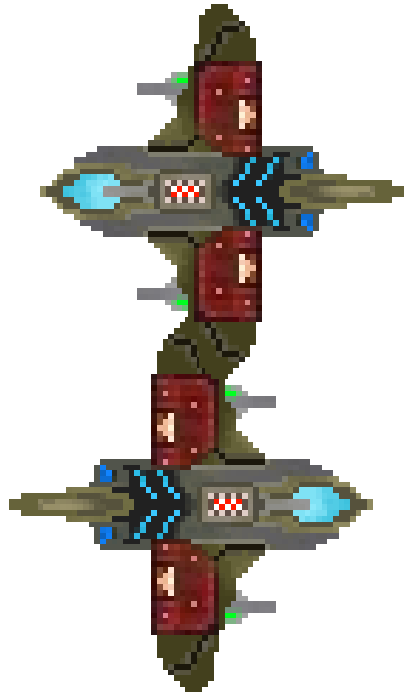
Kollisionshantering

Lösning 2: Pixel Perfect Collision



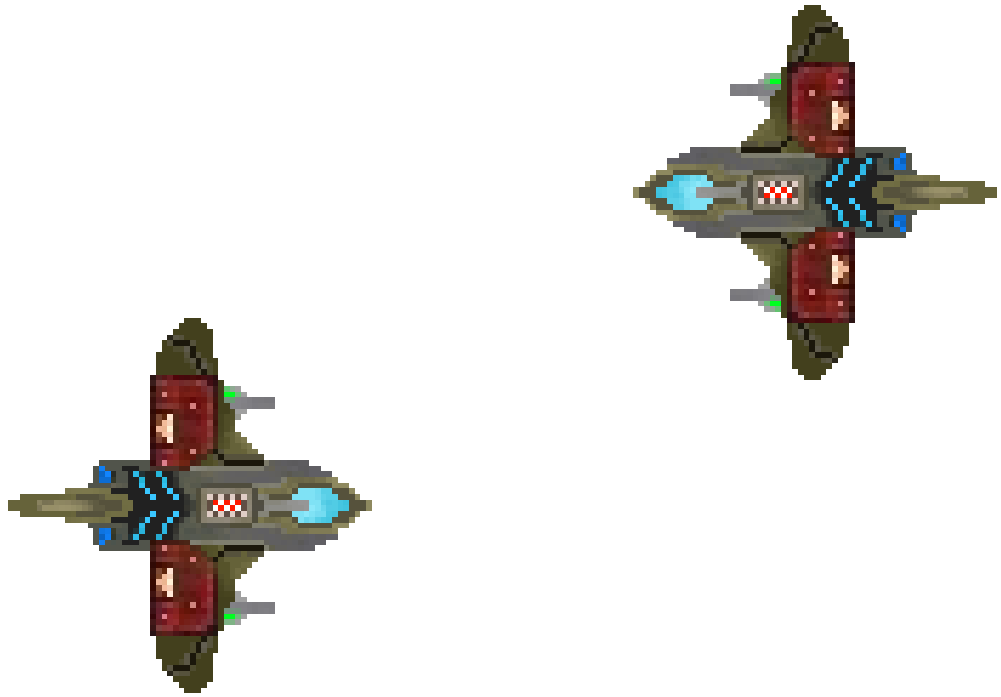
Kollisionshantering

Lösning 2: Pixel Perfect Collision



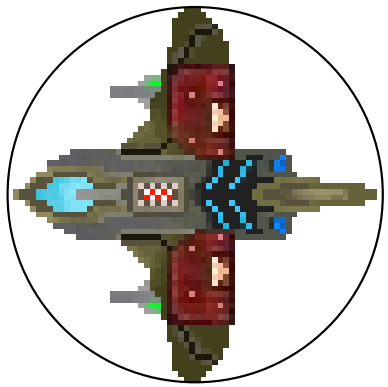
Kollisionshantering

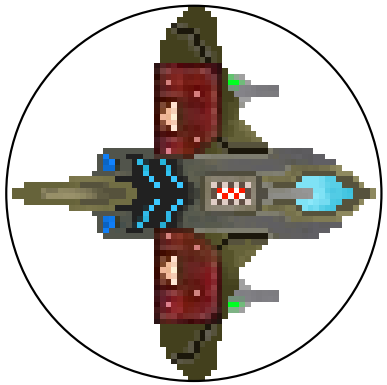
Lösning 3: Radie



Kollisionshantering

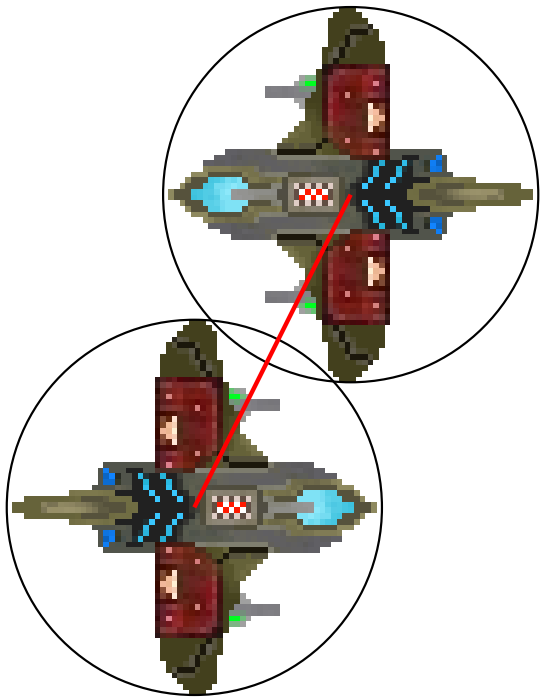
Lösning 3: Radie





Kollisionshantering

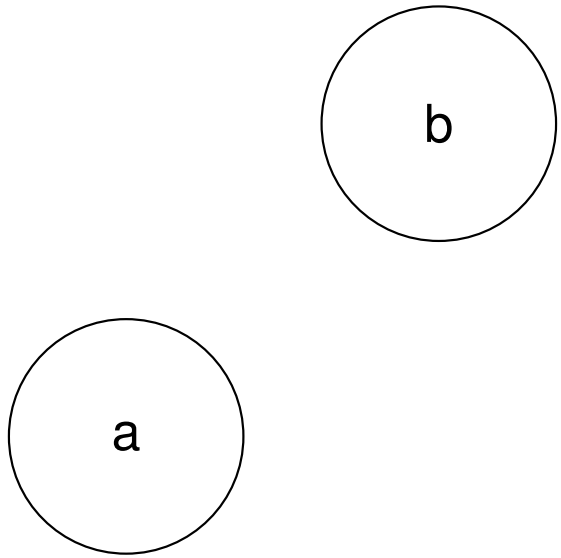
Lösning 3: Radie



Kollisionshantering

Lösning 3: Radie

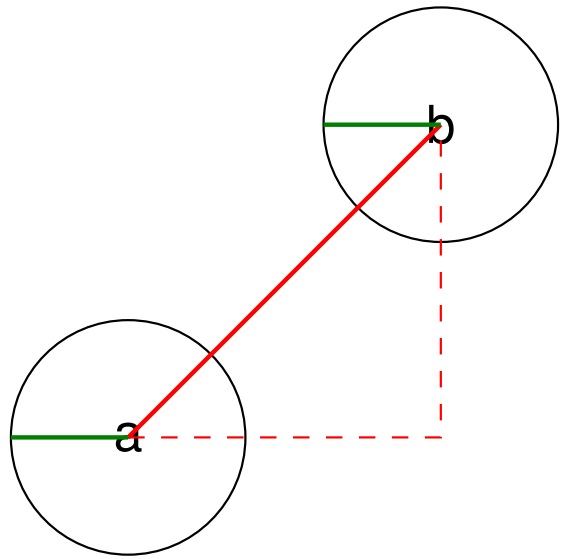
```
struct Radius
{
  int x;
  int y;
  int r;
};

bool collide(Radius a, Radius b)
{
  double d { pow(a.x - b.x, 2) +
             pow(a.y - b.y, 2) };

  return d < pow(a.r + b.r, 2);
}
```



Kollisionshantering

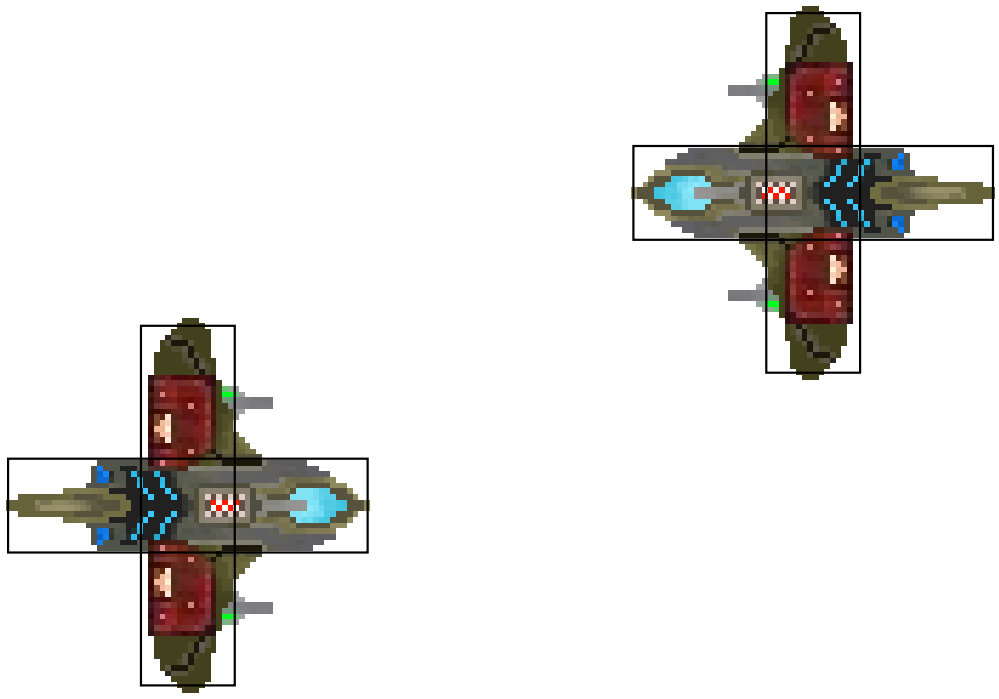
Lösning 3: Radie

```
struct Radius
{
  int x;
  int y;
  int r;
};

bool collide(Radius a, Radius b)
{
  double d { pow(a.x - b.x, 2) +
             pow(a.y - b.y, 2) };

  return d < pow(a.r + b.r, 2);
}
```



Kollisionshantering

Flera AABB



Kollisionshantering

Nästa problem: Hur strukturerar vi vår kollisionsdetektion?

Kollisionshantering

Nästa problem: Hur strukturerar vi vår kollisionsdetektion?

```
vector<object> objs {};

for (auto a { objs.begin() }; a != objs.end(); ++a)
{
  for (auto b { a + 1 }; b != objs.end(); ++b)
  {
    if (collide(*a, *b))
    {
      // objekten har kolliderat!

    }
  }
}
```

Kollisionshantering

Generalliserad lösning:

```
vector<object> objs {};

for (auto a { objs.begin() }; a != objs.end(); ++a)
{
  for (auto b { a + 1 }; b != objs.end(); ++b)
  {
    if (collide(*a, *b))
    {
      a->handle_collision(*b);
      b->handle_collision(*a);
    }
  }
}
```

Låt varje objekt själv avgöra vad som ska ske vid en kollision, och skicka med vilket objekt den kolliderade med.

1 Spelutveckling
2 Tillståndsmaskin
3 Resurshantering
4 Kollisionshantering
5 SFML
6 SFML Demo

SFML

- Står för Simple Fast Media Library
- Används för fönsterhantering, grafik, ljud, tangentbordshantering, nätverk, m.m.
- Hemsida: https://sfml-dev.org
- Finns installerat på IDA:s system

Måste ha med flaggor vid kompilering:

```
$ g++ main.cc -lsfml-window -lsfml-graphics -lsfml-system
```

SFML

Nytt för 2025

- SFML 3.0 släppt
- Vi använder 2.5.1 tills vidare
- Kolla i rätt dokumentation!

1 Spelutveckling
2 Tillståndsmaskin
3 Resurshantering
4 Kollisionshantering
5 SFML
6 SFML Demo

www.ida.liu.se/~TDDI82/