Temas Tratados en el Trabajo Práctico 1

- Diferencia entre Inteligencia e Inteligencia Artificial.
- Concepto de omnisciencia, aprendizaje y autonomía.
- Definición de Agente y sus características. Clasificación de Agentes según su estructura.
- Identificación y categorización del Entorno de Trabajo en tabla REAS.
- Caracterización del Entorno de Trabajo.

Anotaciones

"Acordarse de la definición de agente"

Ejercicios Teóricos

1. Defina con sus propias palabras inteligencia natural, inteligencia artificial y agente.

Inteligencia Natural: es propia de los seres vivos(humanos, animales, plantas). Los seres vivos nacen con esta inteligencia, se basa en emociones, experiencia. Para ser caracterizado como inteligente tiene que ser útil y tener un obejtivo.

Inteligencia Artificial: Es algo sintético, desarrollado por el hombre.

Agente: Entidad que lee u obtiene la información útil para llevar su cabo su objetivo. Son entes con un objetivo específico.

2. ¿Qué es un agente racional?

Busca optimizar el proceso en función al rendimiento buscado.

3. ¿Un agente es siempre una computadora?

No siempre, no todas las soluciones a problemas implican el uso de programas o computadoras. Para cumplir un objetivo un agente puede constar de mecanismos u otras soluciones más simples. Simpre se debe buscar la herramienta para cada problema

4. Defina Omnisciencia, Aprendizaje y Autonomía.

Omnisciencia: idealmente, un sistema conoce toda la información del entorno para desarrollar su función. Aprendizaje: parámetros internos que pueden mejorar en base a

experiencia, así optimizar su rendimiento Autonomía: No necesita supervisión externa para desempeñar correctamente su tarea, teniendo en cuenta el rendimiento buscado. (Agente que pueden desarrollar su propios modelos, donde configura o mejora sus parámetros).

5. Defina cada tipo de agente en función de su **estructura** y dé un ejemplo de cada categoría.

Agente Reactivo Simple: agentes simples con inteligencia limitada. Ej: aspiradora, pava eléctrica Agente Reactivo Basado en Modelos: Agente Basado en Objetivos: se limita a cumplir con la tarea específica. Ej: buscar una ruta de un punto al otro Agente Basado en Utilidad: tiene en cuenta la forma en que debe desarrollar su objetivo. Ej: buscar el camino mas corto o cómodo de un punto al otro.

6. Para los siguientes entornos de trabajo indique sus **propiedades**:

    a. Una partida de ajedrez.

    b. Un partido de baloncesto.

    c. El juego Pacman.

    d. El truco.

    e. Las damas.

    f. El juego tres en raya.

    g. Un jugador de Pokémon Go.

    h. Un robot explorador autónomo de Marte.

a. Partida de agedrez: Totalmente observable - Determinista - Secuencial - Estático - Discreto - Multiagente (Dos jugadores)

b. Un partido de baloncesto: Parcialmente observable - Estocástico - Secuencial - Dinámico - Continuo - Multiagente (Dos equipos)

c. Pacman: Totalmente observable - Estocástico - Secuencial - Dinámico - Discreto - Multiagente

d. El Truco: Parcialmente observable - Estocástico - Secuencial - Estático - Discreto - Multiagente

e. Las Damas: Totalmente observable - Determinista - Secuencial - Estático - Discreto - Multiagente

f. El juego "tres en raya": Totalmente observable - Determinista - Secuencial - Estático - Discreto - Multiagente

g. Un jugador de Pokemon Go: Parcialmente observable - Estocástico - Secuencial - Dinámico - Continuo - Multiagente

h. Robot explorador autónomo en Marte: Parcialmente observable - Estático - Secuencial - Dinámico - Continuo - Un solo agente

7. Elabore una tabla REAS para los siguientes entornos de trabajo:

   a. Crucigrama.

   b. Taxi circulando.

   c. Robot clasificador de piezas.

a. Crucigrama

Rendimiento: Maximizar el número de palabras correctas y completadas en el menor tiempo posible, basandose en descripciones y pistas sobre las palabras que van en los huecos en blanco.

Entorno: Tablero bidimensional de celdas con casillas negras y blancas; definiciones textuales.

Actuadores: Lápiz/teclado para escribir letras en celdas específicas; borrar letras.

Sensores: Vista de las definiciones; vista del tablero y letras ya colocadas; memoria de palabras conocidas.

b. Taxi circulando

Rendimiento: Transportar pasajeros a su destino de manera segura, eficiente y en el menor tiempo, respetando las normas de tráfico.

Entorno: Las calles de la ciudad, los otros vehículos, los peatones, las señales de tráfico, los semáforos y las condiciones meteorológicas.

Actuadores: Acelerador, freno, volante, caja y palanca de cambios, motor, suspensión, ruedas, ejes, luces, mecanismos de cerrojo, bocina, etc.

Sensores: GPS, velocímetro, sensor de combustible, sensor de temperatura, micrófono para comunicarse con el pasajero y camaras de video para tener una vista del entorno (peatones, vehiculos, señales de transito, etc).

c. Robot Clasificador de Piezas

Rendimiento: Clasificar correctamente el 100% de las piezas en su contenedor designado en el menor tiempo posible.

Entorno: Cinta transportadora con piezas de diferentes formas, tamaños o colores.

Actuadores: El brazo robótico que coge y mueve las piezas, y los mecanismos para soltarlas en los contenedores, como pinzas, empujadores o cintas secundarias.

Sensores: Cámaras para identificar las piezas por forma y color, sensores de peso y sensores de posición para detectar dónde están las piezas en la cinta transportadora.

Ejercicios Prácticos

8. La Hormiga de Langton es un agente capaz de modificar el estado de la casilla en la que se encuentra para colorearla o bien de blanco o de negro. Al comenzar, la ubicación de la hormiga es una casilla aleatoria y mira hacia una de las cuatro casillas adyacentes. Si...

- ... la casilla sobre la que está es blanca, cambia el color del cuadrado, gira noventa grados a la derecha y avanza un cuadrado.
- ... la casilla sobre la que está es negra, cambia el color del cuadrado, gira noventa grados a la izquierda y avanza un cuadrado.

  Caracterice el agente con su tabla REAS y las propiedades del entorno para después programarlo en Python:

  ¿Observa que se repite algún patrón? De ser así, ¿a partir de qué iteración?

REAS

1. Agente: Hormiga de Langton.
2. Medidas de Rendimiento: Iteraciones minimas. Seguir las reglas sin errores, Reconocer y modificar el color de las celdas del tablero
3. Entorno: Tablero 2D con celdas blancas y negras.
4. Actuadores: Giro hacia la izquierda y derecha 90 grados, movimiento hacia adelante 1 celda o casilla y modificar color de la casilla.
5. Sensores: Reconocer el color actual de la casilla donde se encuentra.

Propiedades del Entorno

- Observabilidad: Parcialmente observable (La hormiga solo reconoce el color de la casilla actual).
- Determinismo: Determinsta
- Episódico: El comportamiento solo depende del estado de la celda actual.
- Estático
- Discreto

- Agente Individual

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
%matplotlib inline
```

In [7]:
```
import pygame
import random
import sys
import numpy as np
```

In [8]:
```
class LangtonAnt:
    def __init__(self, width=1200, height=900, cell_size=6):
        # Configuración de pantalla
        self.width = width
        self.height = height
        self.cell_size = cell_size
        self.grid_cols = width // cell_size
        self.grid_rows = height // cell_size

        # Inicializar grid con NumPy para mejor rendimiento
        self.grid = np.zeros((self.grid_rows, self.grid_cols), dtype=np.uint8)

        # Inicializar hormiga
        self.reset_ant()

        # Configurar Pygame
        pygame.init()
        self.screen = pygame.display.set_mode((width, height))
        pygame.display.set_caption("Hormiga de Langton - Ubuntu")
        self.font = pygame.font.SysFont('ubuntu', 24)
        self.paused = False
        self.step_count = 0
        self.fps = 60

        # Colores
        self.colors = {
            0: (255, 255, 255),  # Blanco
            1: (0, 0, 0),        # Negro
            'ant': (255, 0, 0),   # Rojo para la hormiga
            'text': (50, 50, 150) # Azul para texto
        }

    def reset_ant(self):
        self.x = random.randint(0, self.grid_cols - 1)
        self.y = random.randint(0, self.grid_rows - 1)
        self.direction = random.randint(0, 3)  # 0=N, 1=E, 2=S, 3=O

    def move(self):
        # Regla 1: Celda blanca
        if self.grid[self.y, self.x] == 0:
            self.grid[self.y, self.x] = 1
            self.direction = (self.direction + 1) % 4  # Giro derecha
        # Regla 2: Celda negra
        else:
            self.grid[self.y, self.x] = 0
            self.direction = (self.direction - 1) % 4  # Giro izquierda
```

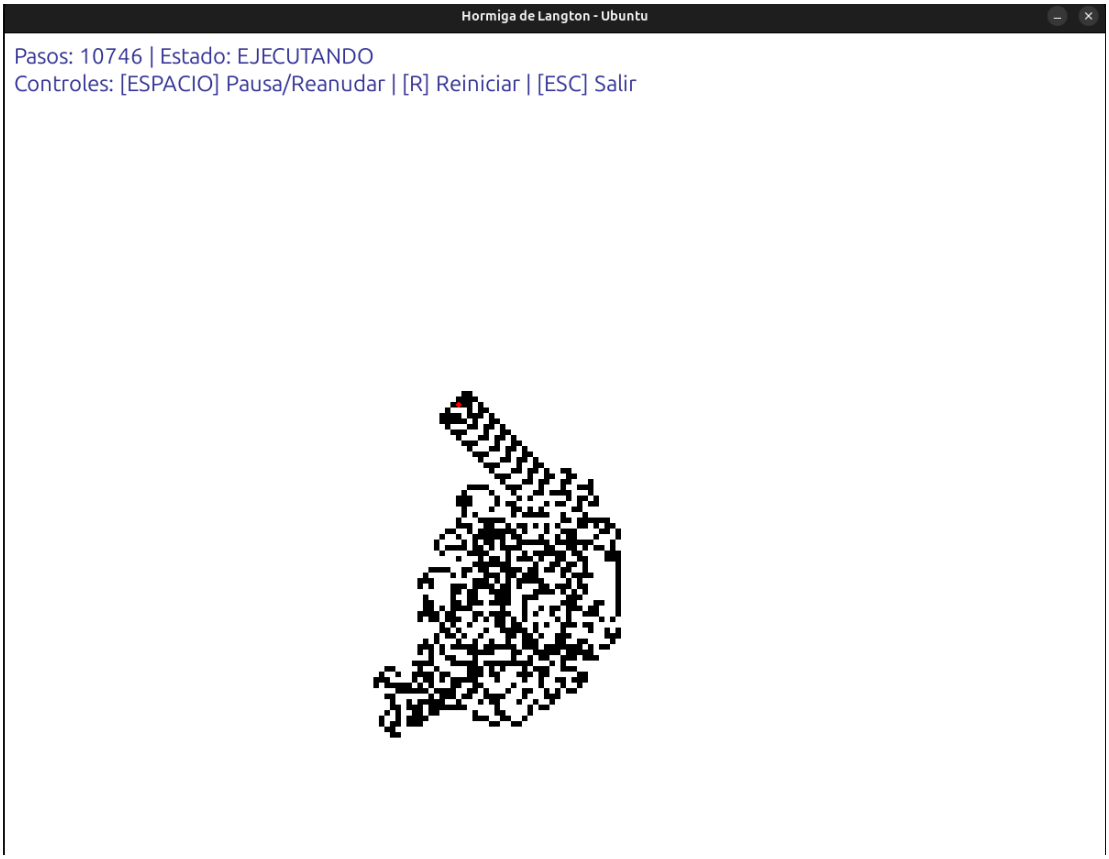
```
        # Avanzar según dirección
        if self.direction == 0:   # Norte
            self.y = (self.y - 1) % self.grid_rows
        elif self.direction == 1:  # Este
            self.x = (self.x + 1) % self.grid_cols
        elif self.direction == 2:  # Sur
            self.y = (self.y + 1) % self.grid_rows
        elif self.direction == 3:  # Oeste
            self.x = (self.x - 1) % self.grid_cols

        self.step_count += 1

    def draw(self):
        # Dibujar fondo blanco
        self.screen.fill(self.colors[0])

        # Dibujar celdas negras
        for y in range(self.grid_rows):
            for x in range(self.grid_cols):
                if self.grid[y, x] == 1:
                    pygame.draw.rect(
                        self.screen,
                        self.colors[1],
                        (x * self.cell_size, y * self.cell_size, self.cell_size,
                    )

        # Dibujar hormiga (más visible)
        ant_size = max(2, self.cell_size // 2)
        pygame.draw.circle(
            self.screen,
            self.colors['ant'],
            (self.x * self.cell_size + self.cell_size//2,
             self.y * self.cell_size + self.cell_size//2),
            ant_size
        )

        # Dibujar información
        status = "PAUSADO" if self.paused else "EJECUTANDO"
        info_text = f"Pasos: {self.step_count} | Estado: {status}"
        controls_text = "Controles: [ESPACIO] Pausa/Reanudar | [R] Reiniciar | [

        text_surface = self.font.render(info_text, True, self.colors['text'])
        controls_surface = self.font.render(controls_text, True, self.colors['te

        self.screen.blit(text_surface, (10, 10))
        self.screen.blit(controls_surface, (10, 40))

    def run(self):
        clock = pygame.time.Clock()

        while True:
            # Manejo de eventos
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()
                elif event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_SPACE:
                        self.paused = not self.paused
```

```
                    elif event.key == pygame.K_r:
                        self.grid = np.zeros((self.grid_rows, self.grid_cols), d
                        self.reset_ant()
                        self.step_count = 0
                    elif event.key == pygame.K_ESCAPE:
                        pygame.quit()
                        sys.exit()
                    elif event.key == pygame.K_UP:
                        self.fps = min(300, self.fps + 10)
                    elif event.key == pygame.K_DOWN:
                        self.fps = max(10, self.fps - 10)

            # Actualizar posición si no está pausado
            if not self.paused:
                self.move()

            # Dibujar
            self.draw()
            pygame.display.flip()
            clock.tick(self.fps)

# Iniciar simulación
if __name__ == "__main__":
    print("Iniciando Hormiga de Langton...")
    print("Configuración recomendada para Ubuntu:")
    print("- Pantalla completa: F11")
    print("- Ajustar velocidad: Flechas ARRIBA/ABAJO")

    # Tamaño optimizado para pantallas de Ubuntu
    sim = LangtonAnt(width=1200, height=900, cell_size=6)
    sim.run()
```

```
Iniciando Hormiga de Langton...
Configuración recomendada para Ubuntu:
- Pantalla completa: F11
- Ajustar velocidad: Flechas ARRIBA/ABAJO
An exception has occurred, use %tb to see the full traceback.

SystemExit
```

Analisis

Observamos que a partir de la Iteración 10000 aproximadamente se empieza a construir una avenida o autopista



9. El Juego de la Vida de Conway consiste en un tablero donde cada casilla representa una célula, de manera que a cada célula le rodean 8 vecinas. Las células tienen dos estados: están *vivas* o *muertas*. En cada iteración, el estado de todas las células se tiene en cuenta para calcular el estado siguiente en simultáneo de acuerdo a las siguientes acciones:

- Nacer: Si una célula muerta tiene exactamente 3 células vecinas vivas, dicha célula pasa a estar viva.
- Morir: Una célula viva puede morir sobrepoblación cuando tiene más de tres vecinos alrededor o por aislamiento si tiene solo un vecino o ninguno.
- Vivir: una célula se mantiene viva si tiene 2 o 3 vecinos a su alrededor.

  Caracterice el agente con su tabla REAS y las propiedades del entorno para después programarlo en Python:

REAS

- Rendimiento: Cumplir las reglas de evolución y actualizar los estados de las células.
- Entorno: Tablero bidimensional con una cantidad finita de células, cada una con dos estados posibles, vivo o muerto. El estado inicial del sistema está definido aleatoriamente.
- Actuadores: Cambiador de estado de las células en cada iteración.
- Sensores: Lectura del estado de cada célula y sus 8 vecinas.

Propiedades del Entorno

- Observabilidad: Totalmente Observable
- Determinismo: Determinista
- Episocidad: Secuencial
- Dinamismo: Estático
- Discreción: Discreto
- Agentes: Un solo agente. Considerando como agente al sistema en su conjunto, encargado de la lectura del estado de todas las celdas y actuando en funcion de sus estados.

In [4]:
```python
class ConwayGame:
    def __init__(self, width=800, height=600, cell_size=10):
        # Configuración de pantalla
        self.width = width
        self.height = height
        self.cell_size = cell_size

        # Calcular dimensiones del grid
        self.grid_cols = width // cell_size
        self.grid_rows = height // cell_size

        # Inicializar grid
        self.grid = np.zeros((self.grid_rows, self.grid_cols), dtype=bool)
        self.next_grid = np.zeros((self.grid_rows, self.grid_cols), dtype=bool)

        # Estado del juego
        self.running = False
        self.generation = 0
        self.fps = 10

        # Configurar Pygame
        pygame.init()
        self.screen = pygame.display.set_mode((width, height))
        pygame.display.set_caption("Juego de la Vida de Conway")
        self.font = pygame.font.SysFont('ubuntu', 20)

        # Colores
        self.colors = {
            'background': (30, 30, 40),
            'grid_lines': (60, 60, 70),
            'cell': (100, 200, 150),
            'text': (220, 220, 240),
            'button': (70, 130, 180),
            'button_hover': (90, 150, 200),
        }

        # Botones
        self.buttons = {
            'start_stop': pygame.Rect(20, 20, 120, 40),
            'clear': pygame.Rect(160, 20, 120, 40),
            'random': pygame.Rect(300, 20, 120, 40),
        }

    def randomize_grid(self, density=0.3):
        """Llena el grid con células aleatorias"""
        for y in range(self.grid_rows):
```

```
            for x in range(self.grid_cols):
                self.grid[y][x] = random.random() < density
        self.generation = 0

    def clear_grid(self):
        """Limpia el grid completamente"""
        self.grid = np.zeros((self.grid_rows, self.grid_cols), dtype=bool)
        self.generation = 0

    def count_neighbors(self, x, y):
        """Cuenta el número de vecinos vivos para una célula"""
        count = 0
        for i in range(-1, 2):
            for j in range(-1, 2):
                if i == 0 and j == 0:
                    continue

                # Coordenadas con condiciones de frontera toroidal
                nx = (x + j) % self.grid_cols
                ny = (y + i) % self.grid_rows

                count += int(self.grid[ny][nx])
        return count

    def update(self):
        """Actualiza el estado del juego a la siguiente generación"""
        if not self.running:
            return

        # Calcular la siguiente generación
        for y in range(self.grid_rows):
            for x in range(self.grid_cols):
                neighbors = self.count_neighbors(x, y)
                current_state = self.grid[y][x]

                # Aplicar las reglas del Juego de la Vida
                if current_state:
                    # Célula viva: sobrevive con 2-3 vecinos, muere por soledad
                    self.next_grid[y][x] = neighbors in [2, 3]
                else:
                    # Célula muerta: nace si tiene exactamente 3 vecinos
                    self.next_grid[y][x] = neighbors == 3

        # Actualizar el grid
        self.grid, self.next_grid = self.next_grid, self.grid
        self.generation += 1

    def toggle_cell(self, x, y):
        """Cambia el estado de una célula específica"""
        if 0 <= x < self.grid_cols and 0 <= y < self.grid_rows:
            self.grid[y][x] = not self.grid[y][x]

    def draw_grid(self):
        """Dibuja el grid y las células"""
        # Dibujar fondo
        self.screen.fill(self.colors['background'])

        # Dibujar líneas de la cuadrícula
        for x in range(0, self.width, self.cell_size):
            pygame.draw.line(self.screen, self.colors['grid_lines'], (x, 0), (x,
```

```
        for y in range(0, self.height, self.cell_size):
            pygame.draw.line(self.screen, self.colors['grid_lines'], (0, y), (se

        # Dibujar células vivas
        for y in range(self.grid_rows):
            for x in range(self.grid_cols):
                if self.grid[y][x]:
                    pygame.draw.rect(
                        self.screen,
                        self.colors['cell'],
                        (x * self.cell_size + 1, y * self.cell_size + 1, self.ce
                    )

    def draw_ui(self):
        """Dibuja la interfaz de usuario"""
        # Dibujar botones
        for button_name, button_rect in self.buttons.items():
            # Comprobar si el mouse está sobre el botón
            mouse_pos = pygame.mouse.get_pos()
            hover = button_rect.collidepoint(mouse_pos)

            # Elegir color del botón
            color = self.colors['button_hover'] if hover else self.colors['butto

            # Dibujar botón
            pygame.draw.rect(self.screen, color, button_rect, border_radius=5)
            pygame.draw.rect(self.screen, self.colors['text'], button_rect, 2, b

            # Texto del botón
            text = ""
            if button_name == 'start_stop':
                text = "Pausar" if self.running else "Iniciar"
            elif button_name == 'clear':
                text = "Limpiar"
            elif button_name == 'random':
                text = "Aleatorio"

            text_surface = self.font.render(text, True, self.colors['text'])
            text_rect = text_surface.get_rect(center=button_rect.center)
            self.screen.blit(text_surface, text_rect)

        # Dibujar información
        status = "EJECUTANDO" if self.running else "PAUSADO"
        info_text = f"Generación: {self.generation} | Estado: {status}"
        text_surface = self.font.render(info_text, True, self.colors['text'])
        self.screen.blit(text_surface, (450, 30))

        # Instrucciones
        instructions = "Controles: [CLIC] Toggle célula | [R] Aleatorio | [C] Li
        text_surface = self.font.render(instructions, True, self.colors['text'])
        self.screen.blit(text_surface, (20, self.height - 30))

    def handle_events(self):
        """Maneja los eventos de Pygame"""
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

            elif event.type == pygame.MOUSEBUTTONDOWN:
```

```
                # Botones de la interfaz
                mouse_pos = pygame.mouse.get_pos()
                if self.buttons['start_stop'].collidepoint(mouse_pos):
                    self.running = not self.running
                elif self.buttons['clear'].collidepoint(mouse_pos):
                    self.clear_grid()
                elif self.buttons['random'].collidepoint(mouse_pos):
                    self.randomize_grid()

                # Toggle de células en el grid
                elif event.button == 1:  # Clic izquierdo
                    x, y = event.pos
                    grid_x = x // self.cell_size
                    grid_y = y // self.cell_size
                    self.toggle_cell(grid_x, grid_y)

            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE:
                    self.running = not self.running
                elif event.key == pygame.K_r:
                    self.randomize_grid()
                elif event.key == pygame.K_c:
                    self.clear_grid()
                elif event.key == pygame.K_UP:
                    self.fps = min(60, self.fps + 2)
                elif event.key == pygame.K_DOWN:
                    self.fps = max(1, self.fps - 2)

    def run(self):
        """Bucle principal del juego"""
        clock = pygame.time.Clock()

        # Estado inicial aleatorio
        self.randomize_grid(density=0.2)

        print("Controles:")
        print("  - ESPACIO: Pausar/Reanudar")
        print("  - CLIC IZQUIERDO: Activar/Desactivar célula")
        print("  - R: Generar estado aleatorio")
        print("  - C: Limpiar tablero")
        print("  - FLECHAS ARRIBA/ABAJO: Aumentar/Disminuir velocidad")

        while True:
            self.handle_events()
            self.update()
            self.draw_grid()
            self.draw_ui()

            pygame.display.flip()
            clock.tick(self.fps)

# Iniciar el juego
if __name__ == "__main__":
    game = ConwayGame(width=1000, height=700, cell_size=12)
    game.run()
```

```
Controles:
  - ESPACIO: Pausar/Reanudar
  - CLIC IZQUIERDO: Activar/Desactivar célula
  - R: Generar estado aleatorio
  - C: Limpiar tablero
  - FLECHAS ARRIBA/ABAJO: Aumentar/Disminuir velocidad
An exception has occurred, use %tb to see the full traceback.

SystemExit
```

Bibliografía

Russell, S. & Norvig, P. (2004) *Inteligencia Artificial: Un Enfoque Moderno*. Pearson Educación S.A. (2a Ed.) Madrid, España

Poole, D. & Mackworth, A. (2023) *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press (3a Ed.) Vancouver, Canada