

École polytechnique de Louvain

Optimizing radiation therapy dose planification using deep reinforcement learning

Author: **Grégoire MOREAU**

Supervisor: **Benoît MACQ**

Readers: **Paul DESBORDES, Vincent FRANÇOIS-LAVET, John LEE**

Academic year 2019–2020

Master [120] in Computer Science and Engineering

Abstract

Optimizing radiation therapy dose planification using deep reinforcement learning

by Grégoire MOREAU

Introduction and goal Radiation therapy treatments are delivered in fractions to slowly destroy the tumour while avoiding severe side effects in surrounding healthy tissues. While conventional treatments deliver fractions of equal dose each day, recent studies have claimed that there could be an advantage in varying the fraction size during the treatment. The goal of this project is to discover such unconventional fractionation schedules by using deep reinforcement learning algorithms agents that will aim to maximise the probability of destroying tumours while minimising the probability of causing severe side effects to surrounding tissues. These agents will be trained in conjunction with a simulation of tumoural development in healthy tissue to find successful treatments.

Materials and methods A simulation of tumoural development in healthy tissue was implemented, based on a 50x50 grid on which healthy cells and cancer cells are modeled individually. The effect of radiation on these cells is simulated by using the LQ model that gives each cell a probability of surviving a certain dose. Deep reinforcement learning agents will receive observations of the simulation in the form of images that show densities of healthy cells and cancer cells on the grid, and will have to choose treatment parameters such as the dose with which to irradiate the tumour or the time between two fractions. They will use the DQN and DDPG algorithms to learn to optimise the sum of the rewards that they receive after each action that they choose. Reward functions were designed to either find treatments that maximise the number of cancer cells killed and minimise the number of healthy cells killed, or to minimise the total dose used in the treatment.

Results The trained agents suggested treatments that achieved much better results on the simulation than a baseline based on a conventional radiation therapy schedule. However, the simulation was implemented to be fast and several effects and phenomena that are important in radiobiology have been omitted from the model. It is therefore likely that the simulation underestimates the damage that could be caused by the treatments suggested by the agents on healthy tissues.

Conclusion The results are promising and show that the approach is worth pursuing. To further this work, a more complex simulation should be used. It should be able to more accurately represent the side effects caused by a treatment on healthy tissues and should be validated using clinical data.

Acknowledgements

I would first like to thank my supervisor, Professor Benoît Macq, for his enthusiasm and ambition for this project from the day we met last year in Montreal until today. I am grateful for the opportunity that he gave me to work 6 weeks in his lab, for his availability throughout the year, for the various advices that he gave me during our meetings, and for the efforts that he made so that I could meet and speak with researchers that helped me in my work.

I would also like to thank Paul Desbordes for his constant help during this project. I appreciated the feedback that he gave me on my work, which helped me direct my efforts in the right direction, and the words of encouragement that he gave me at various points throughout the year.

Finally, I would like to thank Vincent François-Lavet for his patience in helping me understand the complex algorithms that were used in the project, and for giving me advice as to how to effectively use them.

Table of contents

List of abbreviations	vii
1 Introduction	1
1.1 Cancer and radiation therapy	1
1.2 Goal of the project	2
1.3 Plan of the manuscript	3
2 State of the art	4
2.1 Mathematical modeling of cells and tumoural development	4
2.2 Reinforcement Learning in radiation therapy	8
3 Materials and methods	10
3.1 General approach	10
3.2 Simulation	11
3.2.1 Introduction	11
3.2.2 Base of the model	12
3.2.3 Nutrients	12
3.2.4 Cells	14
3.2.5 Radiation	16
3.2.6 Missing from the simulation	19
3.3 Reinforcement Learning	21
3.3.1 Introduction	21
3.3.2 Algorithms	23
3.3.3 DeeR	26
3.3.4 Integration with the simulation	28
4 Results	34
4.1 Performance of the simulation	34
4.1.1 Start of the simulation	34
4.1.2 Without treatment	35
4.2 Training of the agents	37
4.2.1 Baseline	38
4.2.2 DQN - killed	41
4.2.3 DDPG - killed	44
4.2.4 DQN - dose	47
4.2.5 DDPG - dose	50

4.3 Discussion of the results	53
5 Conclusion and future work	55

List of abbreviations

TCP	Tumour control probability
NTCP	Normal tissue complication probability
RT	Radiation therapy
RL	Reinforcement Learning
Gy	Gray, a unit of absorbed radiation (=1 J/kg)
LQ	Linear-Quadratic
MDP	Markov Decision Process
DQN	Deep Q-Networks
DDPG	Deep deterministic policy gradient

Chapter 1

Introduction

1.1 Cancer and radiation therapy

Cancer is a group of diseases caused by an abnormal growth of cells that can spread uncontrollably to other tissues and organs. This abnormal cell growth is caused by an inhibition of the normal cell behaviour that would normally prevent uncontrolled cellular proliferation, potentially through apoptosis. The tumour invasion of the organs leads to the loss of their biological functions, leading to the patient's death.

According to the World Health Organization, cancer was the second cause of death worldwide in 2018 [1]. It will affect an ever larger portion of the population in the future as the number of elderly people increases every year and this group is likelier than average to contract those diseases.

Fortunately, treatments are improving, which leads to higher survival rates and better outcomes for patients. The main types of cancer treatments are surgery, to remove the tumour from the body, chemotherapy, where drugs are used to kill cancer cells, and radiation therapy, where radiation is used to kill cancer cells. These different types of treatments can be used alone or can be combined depending on the situation.

This project will focus on radiation therapy, and in particular External Beam Radiation Therapy or EBRT, as opposed to internal radiation therapy which describes the placement of a source of radiation close to the tumour inside the body. In EBRT, a dose of radiation is sent to the tumour under the form of a beam of photons, electrons or protons. This dose is measured in grays (Gy). One gray corresponds to one joule of absorbed radiation energy per kilogram of matter. The DNA of irradiated cells will be damaged by radiation. This damage, if it cannot be repaired, can cause the cell to die or prevent it from replicating[2].

While the beam goes through the body to irradiate the tumour, some portion of its energy is scattered inside otherwise healthy tissues, which can cause grave side effects. The dose cannot be too high, or the damage done to normal tissue will

negate the positives that come with the destruction of the tumour.

Fortunately, the same mutations that cause cancer cells to evade normal regulatory behaviours also make them more sensitive to radiation than normal cells. They are less capable of repairing single and double strand breaks in their DNA, and tumours are thus slower to recover from small doses of radiation than normal tissues.

To take advantage of this difference of radiosensitivity, the total radiation dose is delivered in fractions that should slowly destroy the tumour while allowing surrounding tissues and organs to repair damages over time. The treatment schedule aims to optimize two criteria :

- The Tumour Control Probability (TCP) is the probability that the tumour is destroyed after the treatment, and should be maximized.
- The Normal Tissue Complication Probability (NTCP) is the probability that tissues and organs close to the tumour will suffer from side effects because of the treatment, and should be minimized.

Currently, at the start of a conventional RT treatment, a total dose to be administered is determined to optimize the TCP and NTCP criteria, and it is delivered in daily fractions of equal dosage of around 2 Gy, as this has proved to provide good clinical results [3].

Recently, studies [3][4] have claimed that more unconventional schedules could yield better results by varying the fraction size during the treatment. To find these better treatments, mathematical models of the tumour can be used with an algorithm controlling the treatment that aims to optimize the TCP and NTCP criteria.

However, this is a complex problem for regular optimizers as each action cannot be optimized independently; the algorithm has to find optimal sequences of fraction sizes. Furthermore, the tumour's development and its reactions to radiation is stochastic. The algorithm should thus be able to take into account all possible developments, and to react to them once they occur.

Reinforcement Learning (RL) is a branch of machine learning in which an agent aims to optimize its sequential decision making by learning from interactions with the environment in which it evolves. A trained RL agent should be able to select actions that lead to optimal long term outcomes and adapt to developments of the environment. RL algorithms are therefore well-suited to find optimal treatment schedules for radiation therapy.

1.2 Goal of the project

The approach used for this master thesis is based on the ideas explored in Jalali-manesh et al.[5]. In this paper, a simulation of the proliferation of cancer cells in healthy tissues was first developed using an agent-modeling program. This simulation includes the evolution of cells in their cycles, their oxygen consumption and

their reaction to a certain dose of radiation. Q-learning, a RL algorithm is then used on top of this simulation to determine the best possible radiation fractionation schedules to treat the developing tumour, using a discretization of the amount of cancer cells into 200 stages of the tumour size as states.

The RL agent could benefit from having more complex information about the current state of the tumour, such as its shape and zones where it is more or less dense in cells. Ideally, this information should also be made similar to scans used by doctors during real radiation therapy, as, in the future, RL agents trained with the same approach could be used to advise doctors if provided with scans of a tumour.

This project will thus extend Jalalimanesh's[5] approach to use more complex information as inputs to the RL agents. To achieve this, modern deep reinforcement learning algorithms will be used instead of simple Q-learning as they are capable of handling higher dimensional inputs. The same articulation was used; a simulation of tumoural development in healthy tissue was implemented and used in conjunction with reinforcement learning algorithms.

1.3 Plan of the manuscript

The next chapter of this manuscript will give a quick review of different approaches used to simulate the behaviour of cancer in humans and its reactions to radiation, and the different ways reinforcement learning has been used to suggest possible improvements to cancer treatment.

In chapter 3, the overall approach of the project will be summarized, and the workings of the simulation and the utilized reinforcement learning algorithms will be explained in detail.

In chapter 4, the result of the training of the agents in different configurations will be exposed and their proposed treatments will be evaluated. This will be followed by a discussion of those results.

Finally, in chapter 5, the manuscript will be concluded by summarizing the main takeaways of this project and suggest ways in which this work could be furthered.

The entirety of the code described in the following chapters, including the source code of the simulation exposed in section 3.2 and the code used to interface the simulation with the agents as well as to train them is available on an open Github repository at the URL https://github.com/gregoire-moreau/radio_rl [6].

Chapter 2

State of the art

2.1 Mathematical modeling of cells and tumoural development

Cell-based computational models can be used to simulate the behaviour of human cells and their possible developments. They are particularly useful in the field of cancer biology as they allow to study the proliferation of these diseases and their reactions to potentially new treatments without resorting to clinical experiments, as those are more expensive and less ethical[7]. These treatment simulations are called *in silico* experiments [3], referencing the silicon based microchips on which they are run, as opposed to *in vivo* and *in vitro* experiments.

Of course, the results of those experiments are only useful if the simulations on which they rely are representative of the real development of tumours in the human body.

Metzcar et al [8] give a general review of the different approaches that can be used to model the development of cancer. Those approaches can be split in two groups:

- Lattice-based methods that place cells on a mesh (or grid)
- Off-lattice methods that allow cells to have free positions

The freedom of position afforded to cells in off-lattice methods typically causes them to be less efficient as it is more computationally intensive to determine which cells are close to which and how they affect each other.

Lattice-based methods can themselves be separated into three categories:

- Cellular automata, where each site (or pixel) on the grid contains a single cell.
- Lattice gas cellular automata, where sites can contain multiple cells.
- Cellular Potts models where multiple sites are used to model a cell including its shape.

The approaches described in the review understandably aim to model the dynamics of tumour cell behaviour in high detail, and are therefore computationally intensive. This makes them relatively hard to use in this project since deep learning algorithms usually require thousands to millions of examples to converge. However, it is still interesting to study those models to determine the most important concepts that need to be included in the project's simulation, and how these concepts are commonly implemented.

Anderson et al.[9] study interactions between tumour cells and their microenvironments, and the roles of those interactions in tumour invasions, using three different models. The first two are hybrid cellular automata, where tumour cells are placed on a discrete grid, while nutrients and chemicals are represented in a continuum. The third aims to model the mechanics of the elastic boundaries of tumour cells in viscous fluid.

The main findings of this article are that tumours growing in microenvironments low on nutrients are more likely to develop finger-like expansions from the main tumour volume. This is verified across all three models. It can be explained by the idea that tumour cells that grow away from the tumour have access to more nutrients and are thus more likely to survive in low nutrient environments.

Smallbone et al. [10] also use a hybrid cellular automaton approach. The model is based around a 2D grid where each pixel can either contain a tumour cell or be empty. The model also keeps track of amounts of oxygen, glucose and H^+ . Under normal conditions, cells produce energy using aerobic respiration, which consumes glucose and oxygen. When oxygen density is low, cells can resort to the less efficient anaerobic respiration, where glucose alone is consumed to produce energy. However, this reaction also releases lactic acid, which is represented in this simulation by the H^+ elements. The acidification of the environment is toxic to cells and can lead to necrosis or apoptosis.

The main result of this paper is that the emergence of tumour cells with phenotypes that make them more resistant to acidity may play a big role in the development from initial growth of a tumour to tissue invasion.

McDougall et al. [11] create a model to study the development of a vascular architecture around the tumour during angiogenesis. Angiogenesis is "the process by which new blood vessels develop from an existing vasculature, through endothelial cell sprouting, proliferation and fusion".

In the case of tumours, angiogenesis happens when the vasculature around a small group of cancer cells is no longer capable of delivering enough nutrients to sustain their development. Cells can react to this lack of nutrients by releasing tumour angiogenic factors in their environments. Endothelial cells, which form the inner layer of blood vessels, are attracted by those factors and can over time develop new vasculature through migration towards the tumour.

The goal of the article is to study this process and to find therapeutic options to prevent it using a simulation. They simulate it by modeling individual endothelial cells, their movements depending on the different factors present in their environment, and the vascular network formed following their migration. Cells are on a grid and can either stay in place, or move up down, left, or right. Each of these possible movements is attributed a probability using equations that take tumour angiogenesis factors into account, and the cells thus move across the grid using a random walk, where the followed path forms the new vascular network. The model is then expanded by adding blood flow through this new network. Figure 2.1 shows the process of adding the blood flow inside the created capillary network until a steady state is reached after around 300 seconds of simulation.

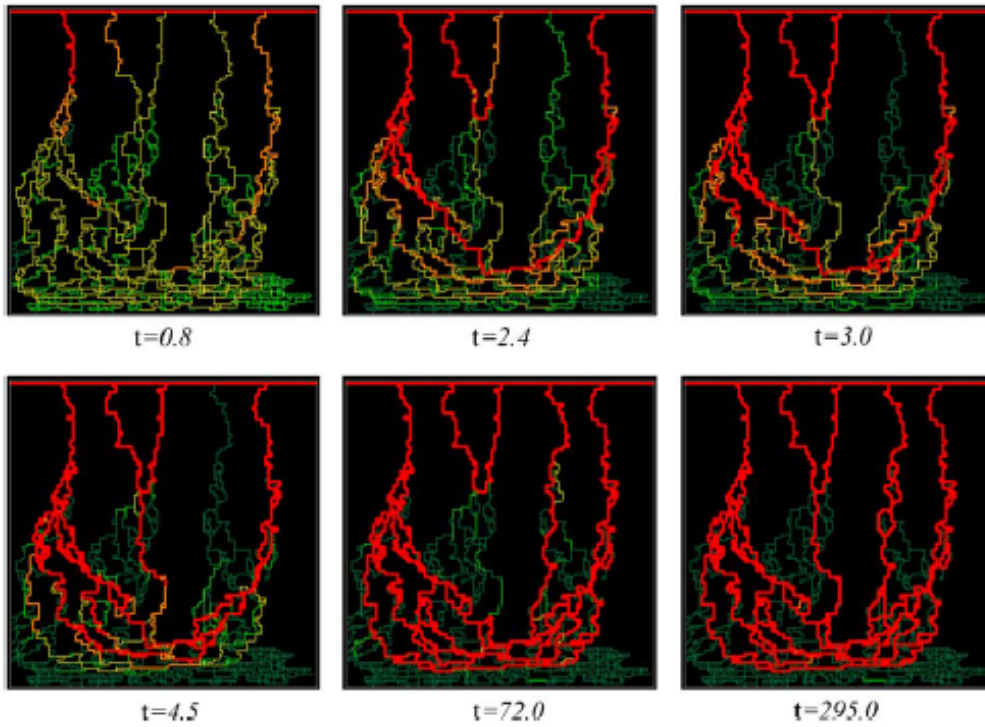


Figure 2.1: Addition of blood flow inside a hollow capillary network [11]

Since these models are too complex and specialized to allow for implementations fast enough to provide the amount of training samples to train a machine learning algorithm, it is necessary to find a simpler model on which the project's simulation can be based.

A first example of a simple simulation is shown in O'Neil [12]. In this work, a simulation of tumoural development in healthy tissue was implemented. In this simulation, healthy cells and cancer cells are represented as agents that proliferate semi-independently¹ on a 2D grid. At the start, a certain amount of healthy cells

¹Cells don't have interactions with other cells but their actions can affect them as they consume nutrients and form density.

are placed randomly on the grid, which also contains an amount of glucose on each pixel.

A single cancer cell is placed at the center of the grid. Over time, the cells will go through the cell cycle and replicate, the grid will be filled almost uniformly by healthy cells, then a tumour will slowly grow around the first cancer cell. If left untreated, the tumour will eventually take over the whole grid while killing the healthy cells.

Radiation is then added to the model to simulate radiation therapy. It is represented by applying a survival probability of 37 % to each cell in a certain radius of a center of radiation chosen by the user of the simulation. The radiation dose is then repeated every 24 hours of simulation.

Cells that are not directly killed by the radiation keep count of the doses that they have received and have to repair the radiation damage over time. Irradiated healthy cells have a probability of mutating during mitosis, which means that they spawn a new cancer cell. Bystander effects are represented by attributing amounts of radiation to cells that were not in the radius of irradiation, depending on the number of irradiated pixels in a small region around them.

The simulation used in Jalalimanesh[5] is largely based on O’Neil’s model. Figure 2.2 shows the growth of an untreated tumour in their implementation.

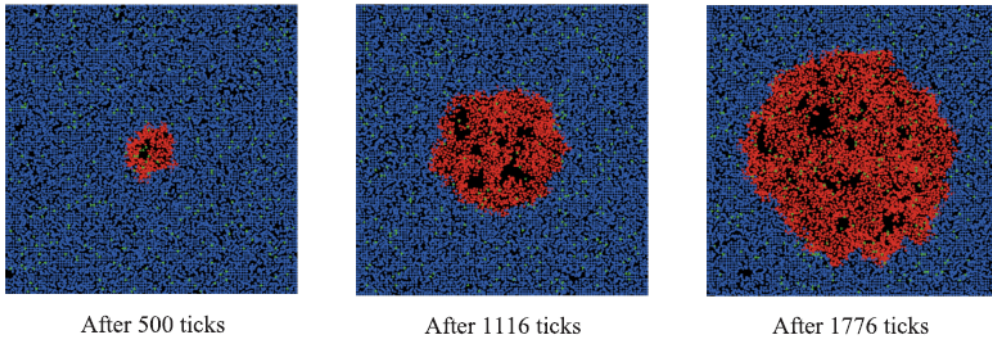


Figure 2.2: Growth of a tumour without treatment in Jalalimanesh[5]

Two important modifications have however been made in order to better model radiation therapy. First, they replace glucose by oxygen as the main nutrient on the grid, as the amount of oxygen available to a cell has an influence on its radiosensitivity.

Second, they use a specific model to compute the survival probability of cells. This model takes into account the dose of irradiation, the oxygen density and the current phase of the cell inside the cell cycle. It is based around the basic linear-quadratic (or LQ) model which aims to describe the surviving fraction of a group of cells subjected to a radiation dose D [13]. It has the form :

$$SF(D) = \exp(-\alpha D - \beta D^2)$$

Where D is the radiation dose in Gy and α and β are parameters corresponding to the radiosensitivity of the irradiated tissue. Van Leeuwen et al. [13] gives an overview of the different values of α and β found in literature. The radiation effect model used by Jalalimanesh [5] is based on an expansion of this model to take into account the evolutions of a cell's radiosensitivity depending on its current phase in the cell replication cycle as well as its level of oxygenation. This model was proposed in Powathil et al.[14] and has the form :

$$S(D) = \exp[\gamma(-\alpha \text{OMF } D - \beta(\text{OMF } D)^2)]$$

Where γ is a parameter representing the effect of the cell's position in the cell cycle and OMF is a factor corresponding to the current oxygenation level. In Powathil [14], instead of simply representing the fraction of surviving cells, the model is used to directly compute the survival probability of each individual cell under a radiation dose. The effect of radiation can thus be implemented by going through each cell in the simulation, applying this model and making it survive or die depending on the predicted survival probability.

In this project, the simulation will be based on the implementations exposed in O'Neil [12] and Jalalimanesh [5] as they are simple enough to allow high rates of training examples. Some biological effects have however been added to the model, as the first articles have shown that they can have big impacts on the developments of cancers and the simulation should be made as realistic as possible despite its efficiency constraints.

2.2 Reinforcement Learning in radiation therapy

Reinforcement learning is a branch of machine learning in which an agent learns to optimize its decision making using observations of its environment and feedback on the actions that it takes [15]. It is capable of efficiently determining optimal sequences of actions and it is therefore increasingly used in the domain of healthcare to improve treatments by choosing at each step the best therapeutic action to reach the best outcome for the patient.

In RL algorithms, the feedback that the agent receives is a scalar called the reward. The agent aims to maximize the sum of rewards over time. It is important to carefully design how those rewards are computed, as this defines the objective of the agent. This is often complex in real-world problems [16], as the agents need to solve multi-objective problems and out-of-the box solutions that are sometimes found by RL agents in toy environments [17] might not be applicable or safe when applied to real situations. In healthcare applications, as a general rule, we can say the the optimal score that the agent can reach should correspond to the best possible outcome for the patient.

Yu et al. [18] present a review of the different ways that reinforcement learning is used in the healthcare sector. They remark that reinforcement learning can be used to improve radiation therapy in part by studying the effect of varying the

fraction size during the treatment, and that studies have shown that there may be an advantage in using non uniform-fractionation schedules for some tissue types.

The article also underlines the fact that it is hard to define a good reward function for radiation therapy as it is a multi-objective problem : the tumour has to be destroyed, possibly fast, but surrounding tissue should be spared as much as possible.

In Tseng et al. [19], a deep reinforcement learning algorithm, DQN, is used to learn dose fraction adaptation in radiotherapy. Based on an initial dataset of 114 patients that received a radiotherapy treatment for lung cancer, a Generative Adversarial Network (GAN) is used to create a larger amount of training data, while a deep neural network is trained on that data to estimate transition probabilities during the treatment and thus create a "Radiotherapy Artificial Environment (RAE)" for the RL agent. Finally a Deep-Q-Network was trained to estimate the value of each possible dose fraction.

The reward function used is based on the idea of maximizing the probability of controlling the tumour while minimizing the probability of complications to the surrounding tissue. Those probabilities are provided by the RAE deep neural network. The article finds that Deep RL is a viable approach to select dose fractions by automatically adapting to the clinical development of the tumour.

In Jalalimanesh [5], the agent uses the tabular Q-learning algorithm. In simple terms, this algorithm aims to compute the "value" or quality of each possible action in each possible state (situation) by averaging the observed values of those actions during the training of the agent. As it is based on a table indexed by actions and states, it is necessary that their domains are discrete, finite, and preferably small in order to reach accurate estimations of the values in a reasonable amount of time.

In this case, the possible actions were 3 possible doses of radiation (2.5, 3 and 3.5 Gy). As observations, the number of tumour cells in the simulations was discretized into 200 stages, with a step between 2 stages corresponding to an increase of 25 cells. As a reward, the agent receives a scalar corresponding to the difference between tumour and healthy cells killed directly by the chosen dose, minus a factor proportional to the dose and the irradiated area.

Chapter 3

Materials and methods

3.1 General approach

In this project, reinforcement learning algorithms were used in conjunction with a simulation of tumoural development in healthy tissue and the reactions of cells to radiation in order to optimize dose planification in radiation therapy.

Figure 3.1 shows the project's different axes and how they interact.

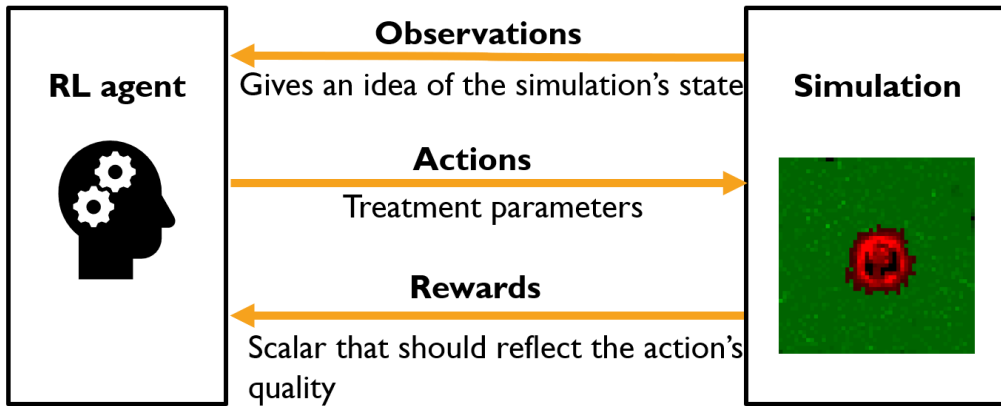


Figure 3.1: Interactions between the agent and the simulation

As described in 2.1, the simulation will be based on the models described in O'Neil [12] and Jalalimanesh [5], with some effects added to make it closer to reality.

The agent will receive observations of this simulation, with which it will be able to understand its current state, choose some parameter to continue the current treatment, for example the dose with which to irradiate the tumour, and will receive a scalar reward indicating the quality of the chosen action.

The agent itself will be trained to optimize its decision making to find optimal treatment plans. To make informed decisions, it needs to have as much information as possible in the observations that it receives.

In Jalalimanesh [5], the observations are based on a discretization of the number of cancer cells present in the simulation into 200 different stages of the tumour size. The environment’s state space is therefore composed of only 200 possible states, which is necessary because the Q-learning algorithm used needs to compute a value for each possible action in each possible state.

The problem with this discretization approach is that the agent might benefit from having more information about the simulation, like the shape of the tumour or its density in different zones. However, the state space resulting from adding all these different features would make training unfeasible in a reasonable amount of time.

Fortunately, new deep reinforcement learning algorithms have been proposed in the last few years that make it possible to tackle the aforementioned problem by using neural networks to translate arrays of raw features into more abstract representations of the environment’s state. In our case, this makes it possible to give the agent images of the tumour and its density so that it can make more informed decisions.

The agent will thus be trained using these deep RL algorithms by repeatedly starting a treatment, choosing its parameters, observing its effects on the simulation using observations and rewards, and slowly improve its decision making. After it has been trained, it should be able to respond to an image in the same format as the observations that it received during training with an action that should lead to an optimal long term score, as defined by its reward function. Therefore, the agent could theoretically be used with more complex and less efficient simulations as long as they develop somewhat similarly to the simulation with which it has been trained.

3.2 Simulation

3.2.1 Introduction

The cellular proliferation model used for this project was based on the master’s thesis of O’Neil [12] and the adaptations made to this model by Jalalimanesh [5].

In this project, implementations of the deep reinforcement learning algorithms that were to be used were available in Python 3. The simulation was therefore first implemented directly in Python, to ensure its compatibility with the algorithms, then as a compiled C++ module importable in Python. This later implementation greatly improved the simulation’s speed, which was necessary as it previously was the bottleneck in the agent’s training.

While most of the parameters that will be described in the following sections have been taken straight from O’Neil’s implementation [12], some of them were adjusted to ensure that a TCP of around 80% was reached for a standard treatment of 2 Gy each day for 35 days.

The entirety of the simulation’s code can be found in the ”model” (for the Python

3 implementation) and "model_cpp" (for the C++ implementation) folders on the project's repository available in open access on Github [6].

3.2.2 Base of the model

The model is based around a two-dimensionnal grid of dimensions 50 by 50. Each pixel on this grid can contain an amount of nutrients, a list of cell objects, and potentially one nutrient source.

The model can thus be described as a lattice gas cellular automaton [8].

The simulation operates in discrete time. A function called `go()` simulates timesteps corresponding to one hour.

3.2.3 Nutrients

Choice of the nutrients

While cells require many different nutrients and chemicals to survive, dioxygen and glucose were chosen to be represented in the cellular model. Those are the two reactants in the summarized chemical equation of aerobic respiration [20] :



The adenosine triphosphate (ATP) molecules produced by this reaction store easily extractable energy in phosphate bonds, and this energy will be used to power the different chemical reactions necessary for the cells' functions.

Oxygen also has a big impact on the radiosensitivity of cells, with hypoxic cells being more resistant to radiation[21].

Glucose has an important role in medical imaging. It can be modified to be used as a radiotracer and injected in the body of a patient. In PET scans, we can observe where the injected tracers are used in the body, which can highlight the presence and position of a tumour as cancer cells have a higher metabolic rate than normal cells and therefore consume more tracers[22].

Nutrient amounts and diffusion

At the start of the simulation, there is a set amount of each nutrient on each pixel. These amounts correspond to 10^{-6} mg of glucose and 10^{-5} ml of dioxygen. These quantities have little influence on the steady state of the simulation; they have been chosen to allow for the early spread of the healthy tissue throughout the grid, and the development of the tumour into a real mass in its centre.

In the model, the nutrients are diffused every hour across the grid. This is implemented by distributing a certain fraction of each pixel's nutrient amounts to its neighbouring pixels. The proportion of nutrients shared to neighbors is called the diffusion rate and is set to 20%.

At the borders of the grid, the pixels also share their nutrients as if they were surrounded by pixels in all directions, and some nutrients are therefore deleted from the simulation after the diffusion. Without this loss, there is a "border effect" where the pixels at the extremities of the grid store large amounts of nutrients, which wouldn't be realistic.

Sources

As each hour, some nutrients are lost at the border of the grid while cells consume large quantities every timestep once they cover a large part of the grid, sources are needed to provide new nutrients.

At the start of the simulation, 100 sources are placed on random pixels of the grid. Every hour, each of these sources will add the equivalent of $1.3 * 10^{-6}$ mg of glucose and $4.86 * 10^{-5}$ ml of dioxygen to the pixel on which they are placed.

In the model, these sources can be considered to represent endothelial cells. In the human body, these cells form the lining of blood vessels in the vascular system, control the passage of materials between these blood vessels and surrounding tissue, and form small vessels around which veins and arteries develop [23].

An important process involving these cells is angiogenesis. As discussed in section 2.1, when tumours reach a stage where their environment can't provide enough nutrients to sustain their growth, they can release Tumour Angiogenic Factors in the extracellular matrix. Endothelial cells will react to these factors by sprouting from their current vessels to create new capillaries in the direction of the tumour. This process will slowly create vasculature around the tumour so that it can continue its growth.[11]

To represent this phenomenon, the sources in the simulation have been allowed to move. Every hour, they have a certain probability of moving from their current pixel. This probability was chosen to be $\frac{1}{24}$, so that the sources move on average once a day, corresponding to a human cell's replication cycle. If the source is selected to move, another probability determines whether the source will move towards the tumour's center of mass, or to a random neighboring pixel. This second probability is computed to be $\frac{N}{50000}$, where N is the number of cancer cells currently forming the tumour. Sources will thus become likelier to converge towards the tumour as it grows and needs more nutrients.

This random walk of the sources first helps to avoid small "islands" of cells surrounding areas containing high concentrations of sources, as these areas aren't constantly in the same place, and ensures that the tumour keeps growing with sufficient nutrients, which is necessary to invade surrounding tissue.

Figure 3.2 shows the densities of glucose and oxygen on the grid after 350 hours of simulation. Lighter colors indicate higher nutrient densities. The bright spots on

single pixels imply the presence of a source on that pixel, while the large zone low on nutrients at the center of the grid corresponds to the tumour.

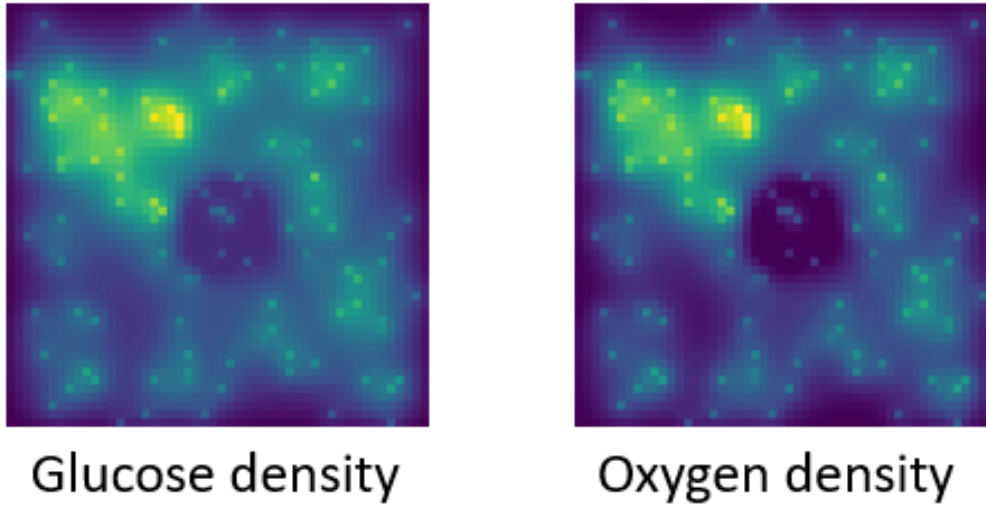


Figure 3.2: Densities of glucose and oxygen on the grid after 350 hours of simulation

3.2.4 Cells

Each pixel on the grid contains a (potentially empty) list of cells.

There are two types of cells in the simulation : healthy cells, representing healthy tissue that would surround a tumour and that should be spared as much as possible from the radiation, and cancer cells, representing the tumour that should be killed by the radiation.

At the start of the simulation, 1000 healthy cells are placed randomly on the grid, and a single cancer cell is placed on its center.

Cell cycle

Every timestep in the simulation, cells will consume nutrients and advance one hour in the cell cycle, or stay or enter in quiescence. Quiescence, also sometimes called the G_0 phase, is a state in which cells stay when they aren't aiming to replicate. Cells can enter or stay in that state due to a lack of nutrients, or because of their genetic programming as a response to environmental factors indicating that proliferation isn't required. In the model, only healthy cells can enter quiescence. They enter or stay in that state if the patch of pixels comprising the pixel on which the cell is placed and its neighbors has an average density of at least one cell per pixel, or if the available amount of nutrients on their pixel sits below a certain level. Cancer cells can not enter quiescence, and will thus keep replicating and invading surrounding tissue.

The human cell cycle modeled in this project is slightly different from the one used by Jalalimanesh et al. The version used here is described in chapter "The Eukaryotic Cell Cycle" from the book "The Cell, a molecular approach" by Geoffrey M. Cooper [24].

A human cell will start its cycle in Gap 1. It will stay around 11 hours in that phase where it simply grows and consumes nutrients. It is in that phase that a cell can enter quiescence and thus stop going through the cell cycle. If it doesn't enter quiescence (or die), a cell will enter synthesis. In this phase, the DNA of the cell is replicated, and this usually lasts about 8 hours. It will then enter Gap 2, during which the cell will keep growing and produce proteins necessary for mitosis. This phase lasts for an average of 4 hours. Finally, the cell will enter mitosis during which it will be divided into two separate cells. This phase takes around an hour. The total human cell cycle thus takes around 24 hours to complete.

In the simulation, the average times in each phase of the cycle are taken to be exact. Healthy cells and cancer cells will thus spend 11 hours in Gap 1, 8 hours in synthesis, 4 hours in Gap 2 and one hour in mitosis, for a total of 24 hours.

During mitosis, the initial cell remains in place and a new cell is created in a neighboring pixel. This neighboring pixel will be chosen randomly among the ones with lowest cell counts for healthy cells' divisions, while it will be chosen randomly among all surrounding pixels for cancer cells' divisions. This ensures that the tumour doesn't spread too quickly across the grid.

Nutrient consumption

In the simulation, not all cells consume the same amounts of nutrients, to reflect real life where the needs of cells depend on their genes and can evolve during their life.

For healthy cells, the amount of nutrients consumed every hour is determined when they are created. A random variable distributed normally with mean 1.0 and standard deviation $\frac{1}{3}$ is sampled to get a real number called the consumption factor. This factor will then be multiplied by the average consumption of a healthy cell to determine its particular hourly consumption of both nutrients, after it has been clipped to be between 0.0 and 2.0 to avoid cells with negative or abnormally large consumptions.

The average hourly consumption of a healthy cell in the simulation is $3.6 * 10^{-9}$ mg of glucose and $2.16 * 10^{-9}$ ml of oxygen.

The same random consumption technique is used for cancer cells, except that the consumption factors are recomputed every hour to show evolutions in the cells needs. As during most of the simulation, the number of cancer cells is an order of magnitude smaller than healthy cells, this doesn't have a big impact on the simulations performance, and it can be particularly interesting to have cells with

differing consumption needs in cases where only a few of them are present on the grid.

The consumption factor for cancer cells is thus also sampled from a normal distribution with mean 1.0 and standard deviation $\frac{1}{3}$. This factor is also clipped between 0.0 and 2.0.

The average hourly consumption of a cancer cell is $5.4 * 10^{-9}$ mg of glucose and $2.16 * 10^{-9}$ ml of oxygen.

In order for a cell to stay alive, the amount of nutrients on its pixel must not be below the critical level of $6.48 * 10^{-8}$ mg of glucose and $3.88 * 10^{-8}$ ml of oxygen.

Finally, healthy cells may enter quiescence if the amount of glucose on their pixel is below $1.73 * 10^{-7}$ mg or if the amount of oxygen is below $1.04 * 10^{-7}$ ml, assuming that both of these quantities are still above the critical level. While in quiescence, healthy cells will only consume 75% of the amounts that they would normally consume while going through the replication cycle.

Figure 3.3 shows how healthy cells and cancer cells are positioned on the grid after 350 hours of simulation, at which time a tumour has developed in the center. Healthy cells are shown in green while cancer cells are shown in red. Lighter colors indicate higher densities.

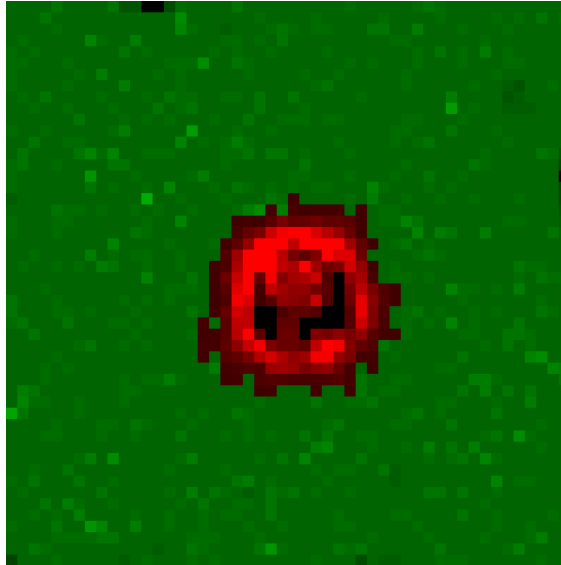


Figure 3.3: Densities of healthy and cancer cells on the grid after 350 hours

3.2.5 Radiation

The effect of radiation on cells is included in the model to simulate radiation therapy.

A radiation dose of a certain amount of grays can be sent to the simulation. First, the current center of mass of the tumour will be determined by taking the average

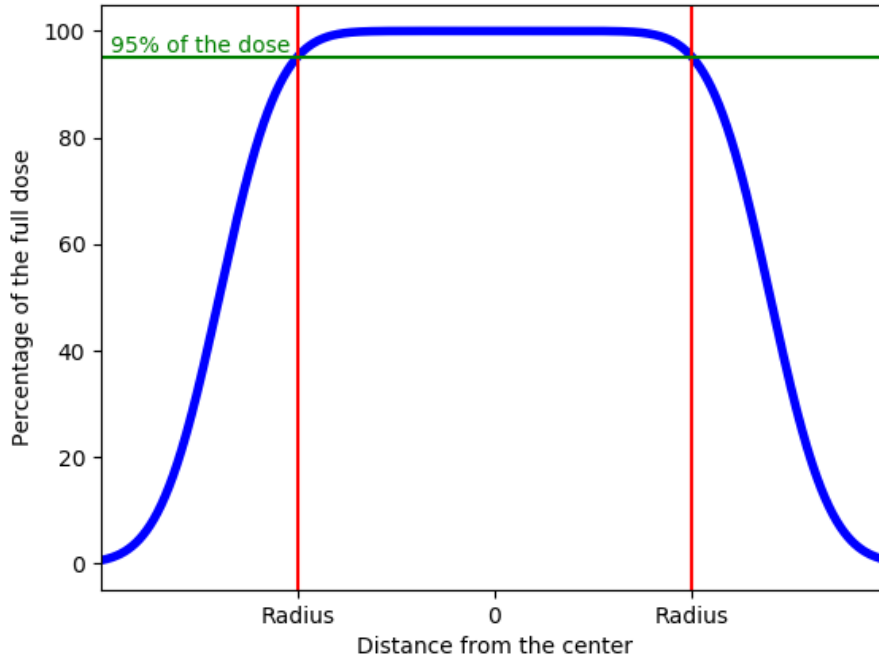


Figure 3.4: Shape of the dose deposition

position of every cancer cell on the grid, and its radius will be computed by finding the distance from this center to the cancer cell furthest away from it.

Using this center and radius, the dose will be distributed on every pixel of the grid according to a specific function that is the convolution product of a rectangular window and a gaussian. The resulting deposition function is shown in figure 3.4. It will be used to determine the dose received by each cell depending on its distance from the tumour's center of mass, as it is also the center of irradiation.

The function's parameters are determined to get 95 % of the agent's chosen gray value on pixels that are exactly one tumour radius away from its center of mass. It can be seen that the function of the dose decreases sharply after that distance and the dose will therefore decrease very fast past the borders of the tumour in the simulation. This preciseness of deposition is meant to represent the accuracy of proton therapy. Proton therapy is a type of external beam radiation therapy in which a beam of protons is used to irradiate a tumour, instead of a beam of photons or electrons. The advantage of this type of treatment is that the energy can be more precisely deposited, as can be seen in figure 3.5 comparing energy deposition for proton therapy and more common approaches of External Beam Radiation Therapy.

In this figure, we can see that protons release most of their energy in a single narrow peak, called the Bragg peak. The depth of this peak in the body can be modulated with the initial velocity of the protons. After this peak, the protons stop almost immediately. In more conventional types of RT, energy is scattered throughout the

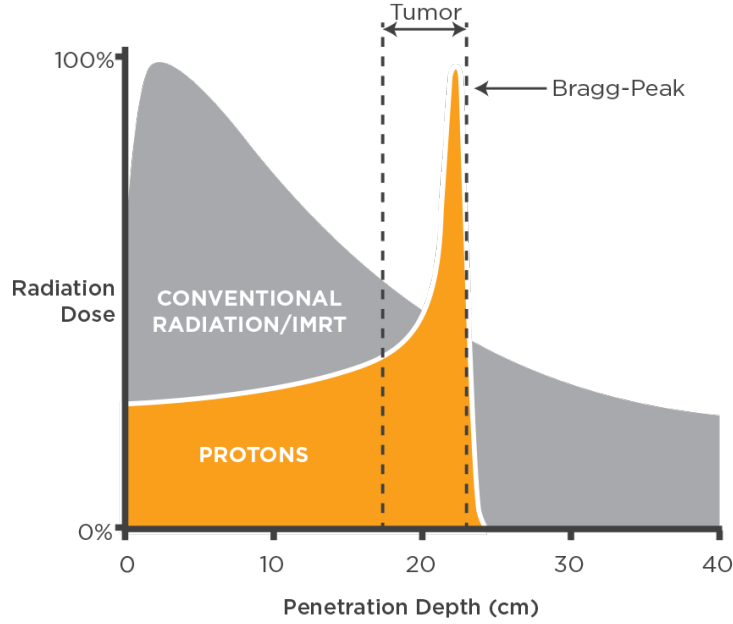


Figure 3.5: Comparison of energy scattering for therapy types, image from [25]

entire path of the photons, and the irradiation is thus less precise.

The effect of radiation is simulated by computing, for each cell, a survival probability depending on the adjusted dose given by the deposition function, the cell's current stage in its cycle, and the amount of oxygen available.

This survival probability is given by the following function, which is a version of the basic LQ model modified in Powathil et al. [14] :

$$S(d) = \exp[\gamma(-\alpha \text{OMF } d - \beta (\text{OMF } d)^2)]$$

In this model, S gives the survival probability of a cell irradiated with a dose d , in Gy.

γ is a factor that reflects the differences in radiosensitivities depending on the current phase of a cell in its cycle. It will have a value of .75 when the cell is in quiescence or synthesis, 1.25 in Gap 2 and mitosis, and 1.0 in Gap 1.

α and β are parameters of the regular LQ model that correspond to the intrinsic radiosensitivity of the irradiated cells [13]. In the simulation, α is 0.3 and β is 0.03 for cancer cells, while α is 0.15 and β is 0.03 for healthy cells.

Finally, OMF is the oxygen modification factor. It is used to represent the fact that hypoxic cell are less sensitive to radiation. OMF can be computed with this formula, again from Powathil [14] :

$$\text{OMF} = \frac{\text{OER}(pO_2)}{\text{OER}_m} = \frac{1}{\text{OER}_m} \frac{\text{OER}_m * pO_2(x) + K_m}{pO_2(x) + K_m}$$

$pO_2(x)$ is the dioxygen concentration in the position of the cell, OER is the ratio of the radiation dose needed to kill the same cell under oxic and anoxic conditions, OER_m is a maximum to this ratio with a value of 3, and K_m is the oxygen concentration that would lead to an OER value of 2, with a value of 3 mm Hg.

Figure 3.6 represents the survival probabilities of a healthy cell and a cancer cell for doses from 0 to 10 Gy using this modified LQ model with values of 1 for γ and OMF.

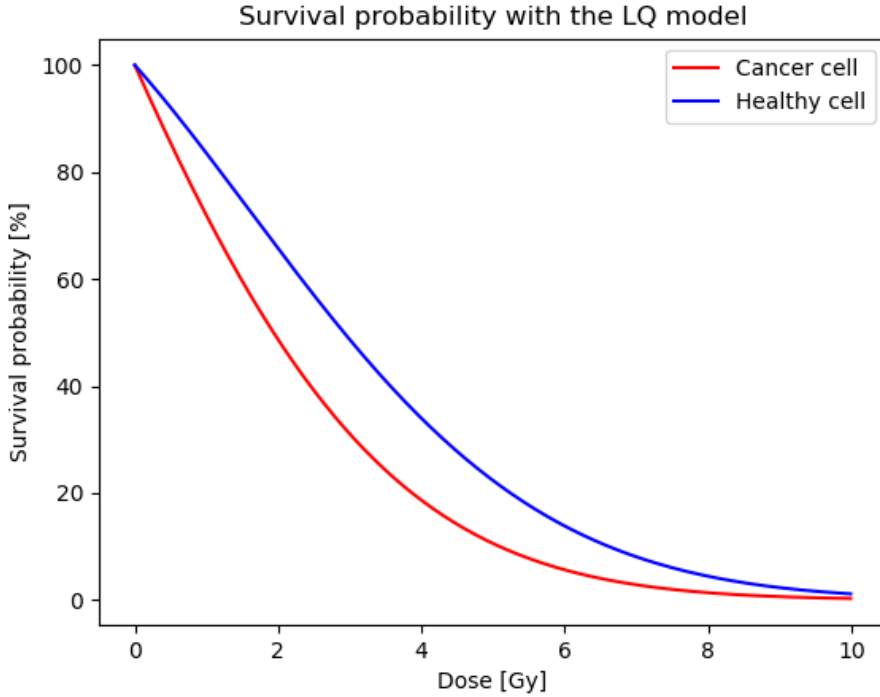


Figure 3.6: Survival probability of a cell as a function of the dose

After obtaining the survival probability of the cell, a random variable distributed uniformly between 0 and 1 is sampled to determine whether the cell is killed by the radiation. If it is, the cell object is simply deleted from the grid. If it isn't, it will have to repair damage from the radiation for some hours. This is needed as otherwise, even with high doses that would lead to survival rates of 50 %, the tumour could still double in size every 24 hours. Cells that have received a dose of more than 0.5 Gy will have to repair for a random time distributed uniformly between 0 and 18 hours.

3.2.6 Missing from the simulation

As explained before, one of the main requirements concerning the simulation is that it had to run fast as the deep learning algorithms used in the project require a large number of examples. In order to fulfil this requirement, some features have not

been implemented in the simulation although they could have improved its realism.

First, cells from a type only differ by the amount of nutrients that they consume. Different phenotypes could have been included, with different behaviours while responding to their environments and to radiation. These phenotypes could have been passed on to new cells created by mitosis and an evolution of the tumour's composition could have been observed during the treatment.

Second, the concept of mutation of healthy cells where a healthy cell would create a cancer cell during mitosis because of irradiation hasn't been included in this simulation, although it was present in O'Neil's implementation [12]. While this is a real phenomenon, it is hard to determine its frequency, and it cause problems when added to the model as these mutations tend to create "second" tumours in the simulation, while the irradiation is only coded to handle all cancer cells as a single tumour.

Third, the nutrient sources are implemented in a very simple manner. A whole vascular network could have been implemented following the path of these sources representing the creation of new blood vessels. It would, however, have been complex to write an efficient implementation of this vascular network that would have also avoided an explosion of the number of sources on the grid.

Fourth, cells are assigned to a pixel when created and don't move during the simulation. Although that is mostly accurate for tissue surrounding a tumour, some types of cancer cells can move and invade healthy tissue, sometimes away from the original tumour in the case of metastasis [1]. This wasn't included in the simulation as since a single phenotype of cancer cells is considered in it, all of the simulation's cancer cells would have had a certain probability of moving, which would have been complex to implement efficiently, as each movement would have entailed a modification of two of the list objects that contain cells on each pixel.

Fifth and finally, the bystander effect wasn't included in the simulation. This effect is observed when a cell that hasn't been directly irradiated enters apoptosis as a result of chemical signals sent in the extracellular matrix by cells that have been irradiated [26]. This phenomenon can sometimes be observed in cells far away from the zone that was irradiated. This wasn't modeled here since, at the start of the treatment, the tumour takes a large portion of the grid, and it is not rare to reach situations where the entirety of the cells on the grid receive a dose of radiation. Since the grid doesn't model cell that could be considered as far from the tumour and the radiation, the effects of radiation on these cells aren't taken into account. It is also difficult to accurately represent this effect without modeling signals travelling in an extracellular matrix.

3.3 Reinforcement Learning

3.3.1 Introduction

Reinforcement learning is one of the main branches of machine learning, along with supervised and unsupervised learning. It proposes a formal framework to solve the problem of sequential decision-making [15]. In this framework, an agent has to interact with its environment by choosing actions to execute in order to maximize a specific metric based on information such as observations of this environment and past actions. This metric is the cumulative sum of rewards, where each reward is a scalar feedback received by the agent after it has taken an action.

The agent usually starts without any knowledge about this environment. Through trial and error, it will aim to learn from its experiences to find increasingly good strategies. This differentiates it from supervised learning, where an agent would learn strategies from labelled examples. It can also be separated from unsupervised learning, as the goal of the agent is not only to understand the structure of the environment.

Formally, a RL problem can be described as a discrete time stochastic control process. In RL algorithms, the problem can be simplified to be a Markov Decision Process or MDP. MDP's are formed by the following elements :

- States s_t , belonging to a state space S
- Actions a_t , belonging to an action space A
- Rewards r_t , belonging to a reward space R
- A transition function T that gives the probability of going from state s to state s' given an action
- A discount factor γ

Using the MDP formalization is useful as it allows for precise theoretical statements[27]. However, it relies on an important assumption.

Indeed, the system is considered to be fully observable through the state representations received by the agent. This means that any possible transformation from an environment state space to an agent observation space is ignored, unlike what is done in more general formalizations with hidden states.

This full observability of the environment allows the problem to be considered as Markovian. This means that the probability of going from one state to another will only depend on the current state and the chosen action, and the probability of receiving a certain reward will only depend on the current state and action. Therefore, the agent doesn't need to look at full histories and can focus on the present.

The agent chooses actions according to its *policy*. This policy is provided, directly or indirectly, by the learning algorithm used by the agent. It maps the observations

that the agent receives from the environment to actions that it should take in the situation.

While the agent receives a reward after each timestep, what it aims to optimize is the sum of rewards over time, so that it can discover strategies that provide good returns in the long term. The *value* of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state[27]. This amount of reward depends on the current policy, and can be mathematically defined with :

$$v_{\pi}(s) \doteq \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| S_t = s \right]$$

In this formula, γ is the discount factor. It is a parameter between 0 and 1 that determines how much the immediate reward should be favored over future rewards.

Setting it to 0 means that the agent will simply maximize the immediate reward, and will thus not be able to learn sequential strategies. Setting it to 1 means that a potential reward in a large numbers of timesteps has as much worth as the reward that will be received by the agent after it chooses its next action. While this corresponds perfectly to the goal of sequential decision-making, it is not tractable for complex environments that run for many timesteps or can even be infinite in duration. It would thus be impossible to accurately estimate values.

Instead, the discount factor is typically set to a number close to 1 so that the agent can estimate values that take into account rewards in a large number of timesteps.

Another value function of interest is the action-value function, defined with :

$$q_{\pi}(s, a) \doteq \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| S_t = s, A_t = a \right]$$

$q_{\pi}(s, a)$ is the return that can be expected from starting in state s , taking action a , and following policy π afterwards [27].

A common problem that needs to be addressed in RL is the exploration/exploitation dilemma. After a few steps of learning, the agent could be tempted to exploit his newfound knowledge by always selecting the actions that provided the best rewards. However, the agent should also keep exploring the possible actions in its environment, to potentially find action sequences that could lead to even bigger rewards. Strategies are thus employed to balance those two objectives.

A popular strategy for this is called the ϵ -greedy algorithm. At each timestep, the learning algorithm used by the agent suggests the action associated with the highest long term value based on its past experiences. With a probability ϵ , the agent will

instead choose a random action in its action space. As long as ϵ isn't 0, every action will thus eventually be explored in every situation.

Although it provides an easy solution to the exploration/exploitation dilemma, the ϵ -greedy algorithm can also cause problems as in some environments the optimal policy requires longer sequences of specific actions to be taken. It is unlikely that the agent will discover such policies if at each timestep there is a relatively high chance of taking a random action instead and thus failing to complete the sequence. Usually, the agent starts with a high value of ϵ that is decreased over time once its estimations of the actions' values are more accurate.

In this project, deep reinforcement learning algorithms will be used. Deep learning is a class of function approximators based on the use of deep neural networks that can be used in machine learning algorithms. These neural networks will allow the agent to learn from high-dimensional observations, such as pictures. This can be used to feed richer representations of the environment's state to the learning algorithm, instead of making empirical decisions to reduce those observations to simpler representations which can sometimes hide the true complexity of the environment.

3.3.2 Algorithms

Deep Q-Networks

This algorithm is based on the basic Q-learning algorithm. It is value-based, which means that before choosing an action, it estimates the value of every possible action in the current state.

The optimal action-value function can be defined as :

$$Q^*(s, a) = \max_{\pi} q_{\pi}(s, a)$$

$Q^*(s, a)$ is the maximum discounted sum of rewards that could be expected under an optimal policy, assuming that the agent has just observed state s and has taken action a . If the agent has access to this function, it can trivially reach optimal decision making by simply choosing action $a^* = \operatorname{argmax}_a Q^*(s, a)$ in each state s .

The goal of Q-learning algorithms is to learn accurate approximations of the optimal action-value function. This approximation is learned iteratively. At the start, the function uses random parameters, then at each training timestep it is updated with :

$$Q_{i+1}(s, a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q_i(s', a') | s, a]$$

As i tends to infinity, Q_i should converge to Q^* . For example, in tabular Q-learning, a 2D array is used to represent this function Q_i . Each entry corresponds to one

state-action pair. At each timestep, the agent observes a state s , chooses an action a (potentially using the array), then receives a reward r and observes state s' . An entry of the array is then updated with this formula :

$$Q[s, a] = (1 - \alpha)Q[s, a] + \alpha(r + \max_{a'} Q[s', a'])$$

where α is the step-size parameter, which can be decreased over time in stationary problems¹ as the estimated average is increasingly accurate.

This approach is easy to implement and useful because it is easy to track which portions of the action- and state-spaces have been sufficiently explored. However, one of its main inconvenients is that it necessarily needs to store $|S| \times |A|$ entries, and sufficiently explore them to get accurate estimations. This means that both the action and state-spaces need to be discrete and small. It is thus necessary to make arbitrary decisions on the representation of observations, which can sometimes simplify the problem at the cost of finding real optimal strategies.

In deep Q-learning, a neural network is instead used to approximate the Q^* function. Observations of the environment are provided as inputs to this network. This means that those observations can be raw data, such as images, and the algorithm will itself learn to translate those inputs into more abstract features.

The network produces as an output the values of the possible actions, depending on the state given as input. A single network is typically used to estimate all the values, instead of training separate networks for each action. This is necessary because the first layers of the network are in charge of deriving abstract features from raw inputs, and should thus be the same for all actions.

To train this network, a loss function needs to be defined, so that the backpropagation algorithm can gradually adjust its parameters to minimize this loss. Unlike supervised learning, the agent doesn't have access to a "correct" value that it should have computed. Instead, a target value needs to be defined. Similarly to tabular Q-Learning, it can be defined recursively :

$$y_t = r_t + \gamma \max_{a'} Q(s', a' | \theta)$$

The loss can then be computed by taking the mean-squared error of the estimations for a batch of instances. Using the same network to compute both the prediction and the target value often makes training unstable. Instead, Mnih et al. [28] use a target network in their 2015 paper "Human-level control through deep reinforcement learning" which introduced the DQN algorithm. This target network is solely used to compute target values, and is simply a cloned version of the main neural network that is updated every C iterations.

¹In nonstationary problems, the values of actions can change over time and α can thus be kept constant to allow estimates to follow this evolution [27]

Another problem that makes deep Q-learning unstable is the fact that neural networks usually make the assumption that training samples are independent, while this is obviously not the case if training batches are simply formed by the last N experiences which are necessarily sequential and thus likely correlated. To solve this, Mnih et al. [28] introduce *experience replay*. Essentially, the last M agent's experiences are stored in a buffer under the form of tuples (s, a, r, s') . M is typically a large number. At each training step, instead of using the last N experiences to form a batch, a random sample of N experiences is selected in the buffer and used to train the network.

The DQN algorithm still requires the action space to be discrete and relatively small, since the network has to predict a value for each possible action at each timestep. It is thus not well suited for high dimensional or continuous action spaces.

During the training of the network, one common problem that can be encountered is that some action's values will be overestimated. This will cause trouble since the max operator will thus tend to favor those overestimated actions.

The ϵ -greedy algorithm is typically used with DQN to balance exploration and exploitation.

Actor-critic learning with Deep Deterministic Policy Gradient

This algorithm is policy-based. This means that instead of trying to estimate the value of each possible action, the agent will learn a function that directly maps each state to the action that the agent should take. To improve this policy over time, it uses a policy gradient. This requires a way to estimate the values of the actions that have been taken, to quantify the quality of the current policy and to move in the direction of policies that select actions with higher values.

To do so, it trains two different deep neural networks, an actor which represents the policy function $\mu(s)$ mapping states to actions and a critic representing the action value function $Q(s, a)$. In their 2016 paper "Continuous control with deep reinforcement learning" [29], Lillicrap et al. show that target networks and experience replay can successfully be used to stabilize the training of such networks.

The critic's goal is thus to approximate the action-value function $Q(s, a)$. The input of the network is an observation and the action that was taken following this observation. Its output is the estimation of the value of taking said action in the observed state under the current policy.

The training of the critic is similar to the training of deep Q-networks. Since there is no correct value to which the estimation can be compared, a target value y_t is computed using a target network with the following expression :

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

Using these target values, the loss can be computed for a batch of N experiences $\{s, a, r, s'\}$ using the expression:

$$L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

The gradient of this loss can then be used to update the critic network's parameters using the backpropagation algorithm.

The actor defines the policy followed by the agent. Its goal is to reach the optimal policy in the environment to maximize the sum of rewards received by the agent over time. The input of the actor network is an observation of the environment's state, and the output is the action corresponding to that observation under the current policy. The actions provided are continuous and can be multidimensional by using one output node for each dimension in the network's structure.

To train the network, a policy gradient is used. It has the following form:

$$\nabla_{\theta^\mu} J = \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

Using gradient ascent with this expression to train the actor network ensures that the policy moves towards higher Q-values.

Although the ϵ -greedy algorithm can be used with DDPG to ensure a balance between exploration and exploitation, it is not necessarily well suited to continuous action domains. Lillicrap et al. [29] use a different strategy, based on adding noise to the actions selected by the actor network.

3.3.3 DeeR

DeeR [30] is a Python library by Vincent François-Lavet that provides implementations of popular deep reinforcement learning algorithms such as DQN and DDPG. It was used in this project to form the agents that seek to optimize their decision-making. The library was fully integrated into the project instead of being simply imported as some small modifications were made to its source code.

To use this library, the user needs to define an object that implements the methods in the Environment interface. This object will act as an interface between the agent and its environment. The main methods that need to be implemented are `inputDimensions()` and `nActions()`, which respectively define the observation and action spaces, `observe()` that returns an observation of the environment's state to

the agent, `act()` that applies the action chosen by the agent to the environment, and `reset()` which puts the environment in a starting state.

The library is centered around the reinforcement learning agent which will be trained to maximize the reward signal received after each called to `act()`. The training of the agent is separated in epochs. Between these training epochs, test epochs can be interleaved. The distinction between these two modes allows to use exploration policies only while training. During training epochs an exploration algorithm such as the ϵ -greedy strategy is typically used. By disabling the use of this policy during testing, it is possible to get a more accurate estimation of the current policy's performance.

In addition to the environment, the agent needs to be provided with a learning algorithm. In this project the algorithms used were DQN and DDPG.

The DQN algorithm is presented in the previous section. In this implementation, the neural networks are implemented using Keras. The network's structure in DeeR is presented in figure 3.7.

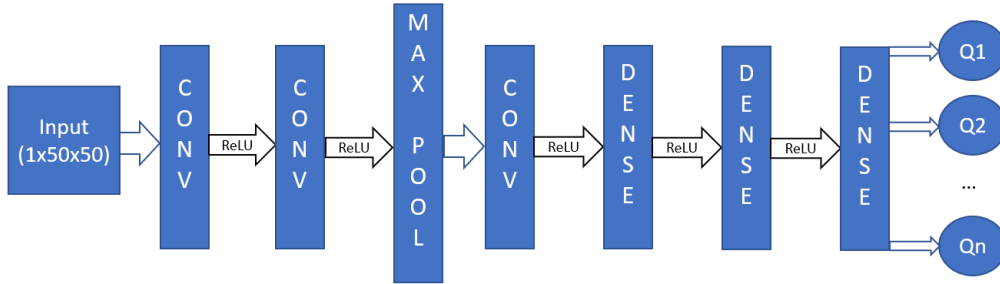


Figure 3.7: Network structure of deep Q-networks in DeeR

The first convolutional layers will serve to detect patterns in the images provided as inputs to the network. Afterwards, the fully connected layers will translate those patterns into abstract features, and those features will finally be used to compute values for each possible action at the output layer.

The DDPG algorithm is also presented in the previous section, and is also implemented in DeeR using Keras. This implementation also uses TensorFlow as backend. The structure of the actor is presented in figure 3.8. The first convolutional layers will detect patterns in the observations received by the agent and will translate these patterns into abstract features. Fully connected (or dense) layers will then use these abstract features to suggest the action that the agent should take in the current state of the environment.

The structure of the critic network used in DDPG in DeeR is shown in figure 3.9. Here, the fully connected layers will use abstract features extracted from the raw

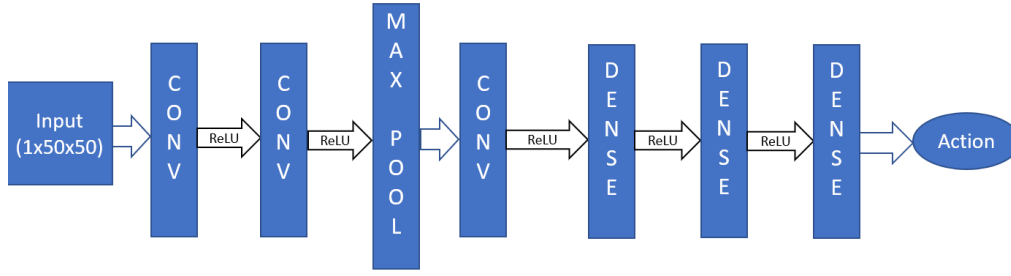


Figure 3.8: Network structure of DDPG actor network in DeeR

observations by the convolutional layers as well as the action taken in the observed state to estimate the value of taking said action in that state.

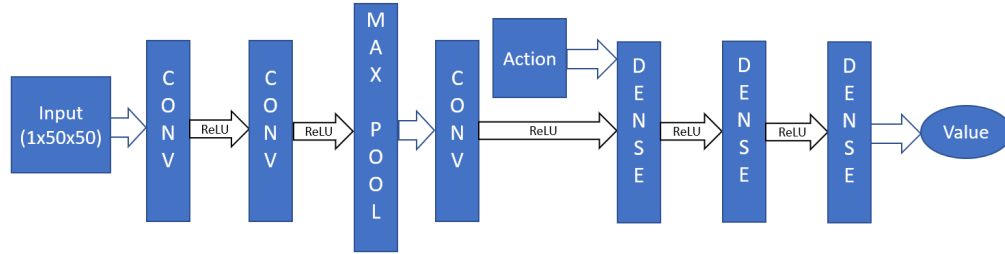


Figure 3.9: Network structure of DDPG critic network in DeeR

DeeR also provides controllers to easily control and update hyperparameters such as the learning rate and the value of ϵ during training.

3.3.4 Integration with the simulation

In order to implement a reinforcement learning agent that learns to suggest a good treatment schedule by using the simulation described in section ??, the different elements of the RL framework have to be defined in terms of the simulation's elements.

It is important to note that the environment in which the agent evolves will not begin at the start of the simulation. Indeed, there is only one cancer cell in the simulation at that time instead of an actual tumour. Instead, the agent will only start to intervene once the tumour has grown to take up significant space on the grid. Currently, the radiation therapy starts at $t = 350$ in the simulation.

Observations

Before choosing an action, the agent receives an observation that allows it to have an idea of the tumour's current state. Experiments were conducted with two different types of observations.

The first is a 50-by-50 image of the grid where each pixel can have three different values. If $S(x, y)$ is the pixel of the observation corresponding to pixel $G(x, y)$ on the grid, then for each position (x, y) :

$$S(x, y) = \begin{cases} -1 & \text{if there is a cancer cell on } G(x, y) \\ 1 & \text{if there are only healthy cells on } G(x, y) \\ 0 & \text{otherwise.} \end{cases}$$

This can be considered to be a segmentation of a scan of the tumour, where each zone is delimited. An example of such a "segmentation" observation is shown in figure 3.10, with black pixels corresponding to values of 0, green pixels corresponding to values of 1 and red pixels corresponding to values of -1.

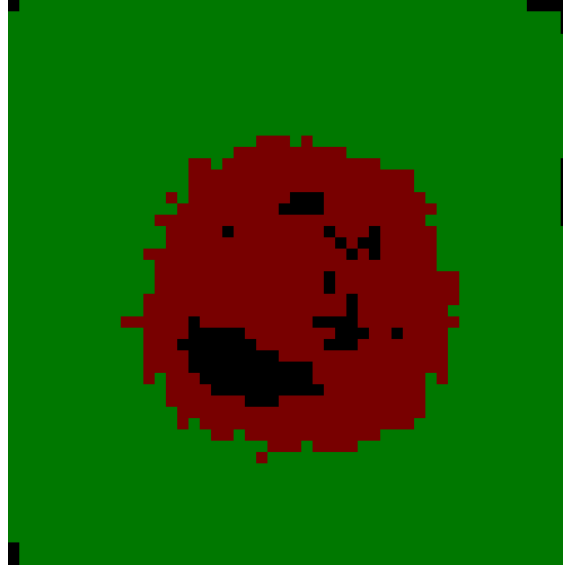


Figure 3.10: Observation of the type "segmentation"

The second observation type is a 50-by-50 image that contains information about cell densities. If $D(x, y)$ is the pixel of the observation corresponding to pixel $G(x, y)$ on the grid, then for each position (x, y) :

$$D(x, y) = \begin{cases} -\text{Number of cancer cells on } G(x, y) & \text{if there is a cancer cell on } G(x, y) \\ \text{Number of healthy cells on } G(x, y) & \text{if there are only healthy cells on } G(x, y) \\ 0 & \text{otherwise.} \end{cases}$$

The additional information brought by the densities can help the agent make more informed decisions about the doses that might be needed and the results that can be expected of those doses. Earlier versions of this type combined all cases of the piecewise definition inside a single sum where each cancer cell would be worth -1 while each healthy cell would be worth 1. However, this would mean that these numbers would cancel out in regions where the tumour has started to invade healthy tissue. This could prevent the agent from having a good idea of the tumour's current

radius, and thus led to worse performances than the current piecewise definition. $D(x, y)$ is divided by 100 on every pixel to guarantee observations between -1 and 1.

In following parts of this manuscript, this type of observations is called "densities" as a reference to the additional information that it brings with the densities of cell types on each pixel. Figure 3.11 shows an example of an observation of this type, where the simulation is in the same state as in figure 3.10. Red pixels indicate negative values while green pixels indicate positive values. Pixels with values of higher absolute values are shown in lighter colors.

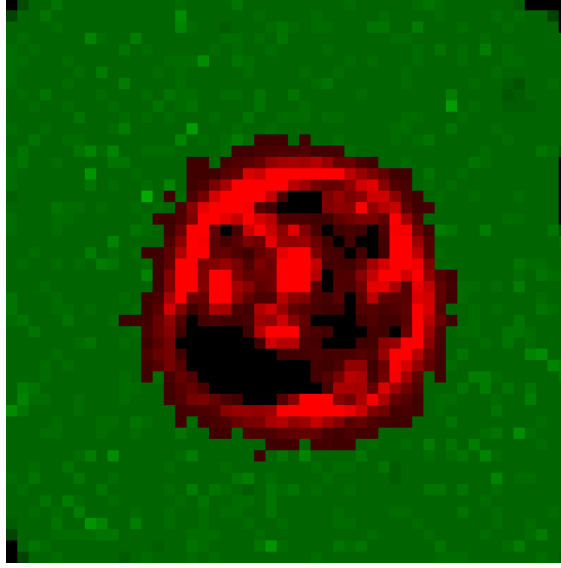


Figure 3.11: Observation of the type "densities"

Actions

As dicussed previously, the agent will act during the treatment by choosing treatment parameters, such as the dose with which the tumour should be irradiated or the rest between two fractions, that should lead to the best possible outcomes for the patient.

In DQN, the action space should be discrete and small, as a value will have to be computed at each step for each possible action. The agents that use the DQN algorithm will choose among 9 possible doses going from 0 to 4 Gy with a step size of .5 Gy. Between each dose, 24 hours will pass as is the case in conventional radiation therapy treatments. A dose of 0 Gy will correspond to an additional rest of 24 hours, if the agent deems it useful.

In DDPG, the action space is continuous and can be multidimensional. Agents that use the DDPG algorithm will be able to choose doses in an interval of $[1, 5]$ Gy and times in an interval of $[12, 72]$ hours of rest after the dose has been delivered. To make it easier for the policy network to output those values, the actual domains in which the agent has to choose are intervals from 0 to 1 and they are scaled back

to be between the limits defined for the dose and the rest times inside the function that applies the chosen actions to the environment.

Termination conditions

The radiation therapy treatment simulated in this project is considered finished in three different situations.

First, the radiation therapy could have been successful. This happens when the tumour has been eliminated. Currently, for that condition to be true, every single cancer cell needs to have been killed through starvation or radiation. This is a harsh condition since doses give survival probabilities that can lead to treatments being extended for tens or hundreds of hours for a few tens of cells remaining. However, one needs to remember that the simulation's tumour was initially formed by a single cancer cell, and that without additional doses, a new tumour will be created from any remaining cancer cells. In real life RT, when the radiation isn't combined with another form of treatment like chemotherapy, the treatment will only be successful once the tumour's cells are in necrosis.

Second, the radiation therapy could have been unsuccessful. The tumour would then have spread too much and have starved almost every single healthy cell in the simulation. At that moment, in the simulation, the tumour would have reached the extremities of the grid and a corresponding real life tumour would thus be out of control with the surrounding tissue invaded. For this condition to be true in the simulation, the number of healthy cells in the simulation needs to drop below 10.

Third, the treatment suggested by the agent could be too long. This situation happens when the agent chooses doses that are high enough to slow the tumour's growth, but aren't high enough to eliminate it. This leads to a sort of status quo where neither the number of healthy cells nor the number of cancer cells are growing. These schedules should be discarded because some nefarious effects aren't included in the model, such as the bystander effect and invasion of tissue by cancer cells breaking away from the tumour. Long treatments chosen by the agent could thus be much more damaging than the simulation predicts, as well as inconvenient for patients. The current implementation considers the episode to be finished after a maximum of 1200 hours or 50 days of treatment.

Rewards

The reward returned to the agent after it has chosen an action should be representative of the quality of the chosen action in the current state of the environment. Furthermore, in cases where there is a clear objective, the reward function should produce values that are higher if they bring the agent closer to its objective.

In RL, a common strategy in environments in which ends either correspond to a success or a failure is to return a single nonzero reward at the end of the episode, for example a 1 for a success and a -1 for a failure.

This is not applicable here, as the agent has two different objectives : destroy

the tumour, and spare surrounding tissue. Furthermore, sparse reward functions² tend to make it harder for the agent to learn as they receive little feedback about individual actions.

It is therefore necessary to create reward functions that indicate to the agent how the chosen action impacted its progress towards these two objectives.

The first reward function that was used to train the agents aimed to achieve both of those objectives. At each regular ³ timestep, the agent will receive a reward :

$$r = (\text{Cancer cells killed} - 5 * \text{Healthy cells killed})/k$$

The number of cancer cells and healthy cells "killed" is a difference between the number of cells before the radiation dose and after the rest that followed the dose. This allows to take into account healthy cells that have been killed by starvation because of the tumour instead of radiation, and gives less of a reward for killing a large number of cancer cells if the tumour has replaced a lot of them through replication in the meantime. The factor of 5, used to give a higher value to healthy cells, was determined empirically as it lead to less aggressive treatments. k is a large number that the values that the agent has to estimate stay between -1 and 1. It was determined empirically to be 100 000.

If this reward function is used for every step, the agent tends to choose policies that allow the tumour to grow large, then sends high doses of radiation in the grid that is now populated by a large number of cancer cells and that will thus produce very large rewards. To avoid this situation, a special reward is given after steps that lead to the end of a treatment. In the case of cancer taking over the whole grid or the treatment taking too much time, a reward of -1 will be returned. If the treatment has however succeeded, the reward will be equal to :

$$r = 0.5 - (Init_{hcells} - End_{hcells})/3000$$

where $Init_{hcells}$ is the number of healthy cells in the simulation at the start of the treatment and End_{hcells} is the number of healthy cells in the simulation at the end of the treatment. This special reward gives an advantage to treatments that allow healthy tissue to slowly reclaim space that was invaded by the tumour during the treatment. 3000 and 0.5 have been empirically found to guarantee that this special reward is between 0 and 1 in all cases.

The second reward function used in this project corresponds to a different approach. RT leads to side effects in normal tissues because they are subjected to high doses of radiation. To find better treatments, one could thus select sequences of actions that lead to successful therapies with the least amount of radiation used. This second reward function will thus aim to minimize the amount of radiation used in the treatment. At each regular timestep, the agent will receive the reward:

²Reward functions that rarely provide nonzero rewards

³that doesn't lead to the end of the episode

$$r = -dose/k$$

where k is a number that ensures that the estimated values stay between -1 and 1. It was determined empirically to be 100.

Trivially, the optimal solution when using solely this reward function is to send the minimal possible dose at each timestep. To avoid this, a special reward is given on steps that lead to the end of an episode where the treatment has failed or lasted too long. The agent will then receive a reward of -1. When the treatment has however succeeded, the agent will receive a regular reward of the form $-dose/k$ to ensure that the optimal treatment is the one corresponding to the minimal dose of radiation.

Chapter 4

Results

4.1 Performance of the simulation

Following the description of the simulation in section 3.2, this section will show images of its evolution and give measures of the computing time needed for this development. While the two implementations (in C++ and in Python) of the simulations correspond to the same mathematical model, they do not run at the same speed. Since the C++ implementation is compiled, it has a large speed advantage, and it was thus used during the training of the agents. The times given here are measured while using the Python interface that interacts with the C++ implementation, and therefore correspond to the times that will be seen by an agent during its training. The times were measured on a computer running Windows 10 and using an Intel Core i7-7700HQ CPU with a frequency of 2.80 GHz.

4.1.1 Start of the simulation

Figure 4.1 shows how the simulation evolves during its first 350 hours, before the treatment starts. Each picture displays the types of cells on each pixel of the grid, with healthy cells shown in green and cancer cells in red. Lighter colors indicate higher densities of these cell types.

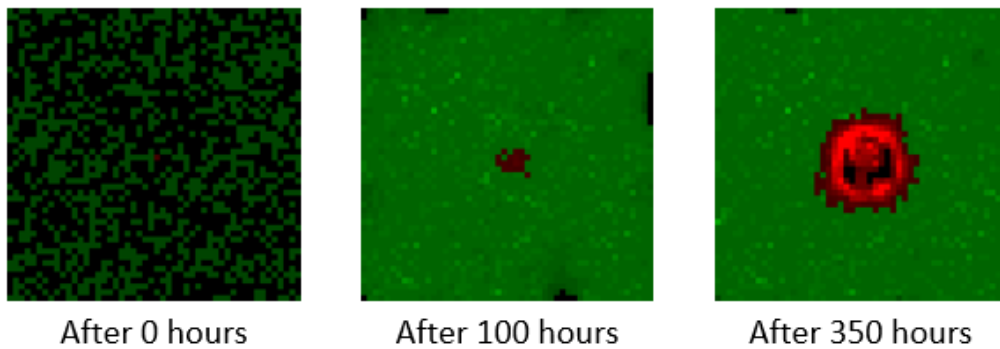


Figure 4.1: Development of cells at the start of the simulation

At the start of the simulation, 1000 healthy cells are placed randomly on the grid, and a single cancer cell is placed in its center. Healthy cells quickly cover the entire grid with almost uniform density while a small tumour starts to develop slowly into a small mass as nutrient amounts can still sustain its exponential growth.

After 350 hours, a large tumour has emerged on the grid. Portions of the tumour's core are already necrotic due to a lack of nutrients. Figure 4.2 shows the evolution of the number of cancer cells on the grid.

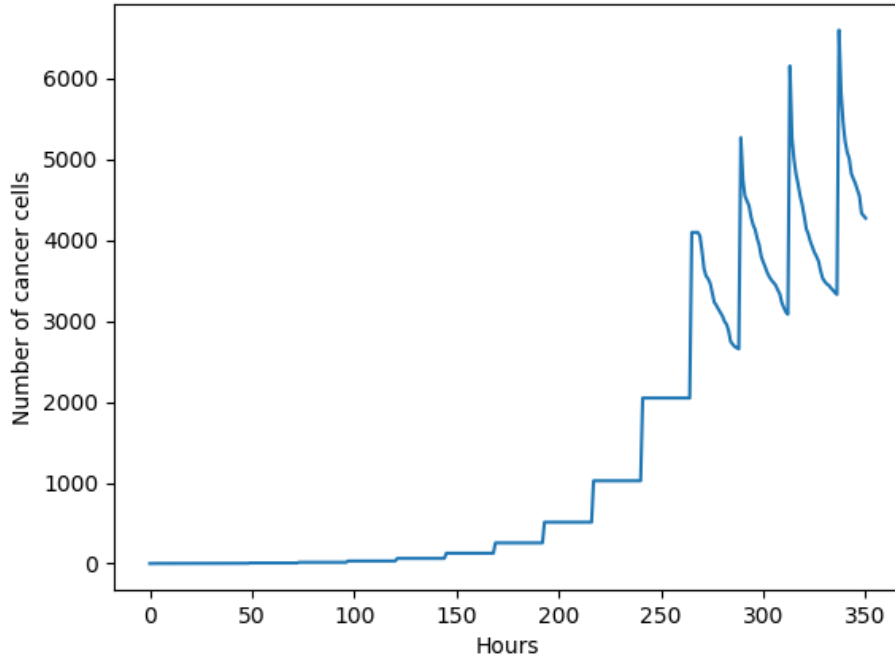


Figure 4.2: Evolution of the number of cancer cells at the start of the simulation

At this time of the simulation, the tumour is formed of around 5000 cancer cells and the radiation therapy will begin. RL agents that aim to optimize radiation therapy treatments in this project will have to reset the environment at the end of each episode in order to create a new tumour on which they can train, and the simulation will thus go through this starting phase at the start of each episode. This reset will take 139 ms on average.

4.1.2 Without treatment

After this starting period, if no treatment is used to destroy or control the tumour, it will quickly grow to take over almost the entire grid. In the meantime, the modeled effect of angiogenesis will cause the nutrient sources to converge towards the tumour's center of mass. This means that surviving healthy cells surrounding the growing tumour are deprived of nutrients, and will thus starve. Figure 4.3 shows this evolution.



Figure 4.3: Growth of the tumour after the starting phase without treatment

In this instance of the simulation, the number of healthy cells drops below ten 985 hours after its start. This relatively short duration (the maximum length of a treatment for the agents is 1200 hours which corresponds to 1550 hours of simulation) can be explained by the effect of angiogenesis that is increased due to the high number of cancer cells in this scenario. Indeed, as can be seen in figure 4.4 which shows the evolution of this number after the starting phase, an oscillation between around 20 000 and 40 000 cancer cells on the grid can be observed. This is significantly higher than the 5000 cancer cells at the end of the starting phase, and in general much higher than the number of cancer cells forming the tumour while some sort of treatment is applied in the simulation.

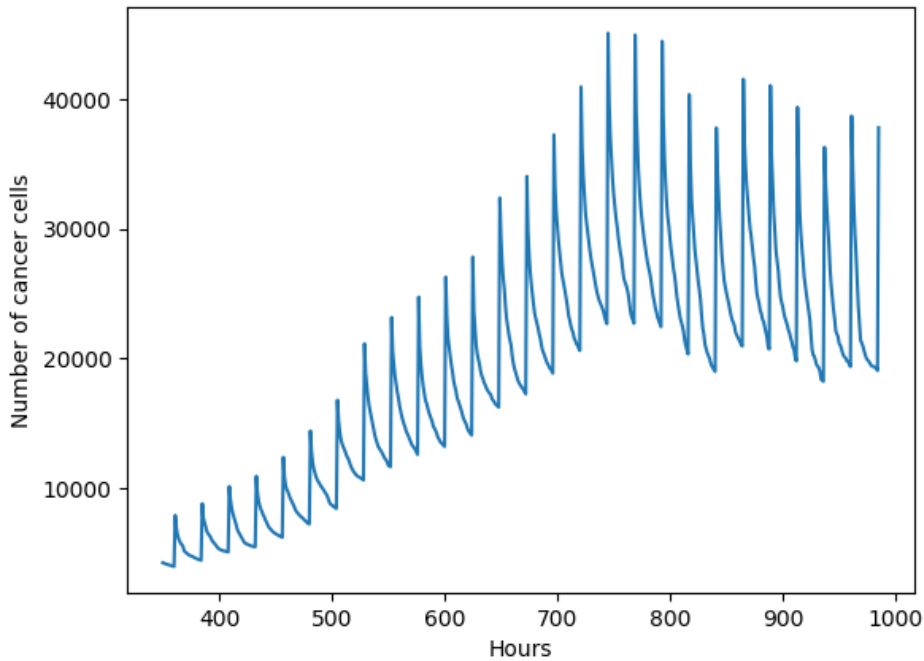


Figure 4.4: Evolution of the number of cancer cells without any treatment

The time that it takes to simulate 24 hours depends largely on the number of cells on the grid, especially since cancer cells have to redetermine their nutrient consumption every hour.

On average, 24 hours simulated right after the starting phase will take 16 ms. In the example shown previously without treatment, the last 24 hours before the number of healthy cells drops below 10 take on average 54 ms.

The evolution of the simulation while a treatment is administered will not be shown here as plenty of examples will be given in the following section.

The time that it takes to irradiate the tumour depends on the number of cancer cells, as well as the size of the tumour. It is however almost¹ independent of the actual radiation dose. The irradiation of a tumour of around 5000 cancer cells such as the one at the end of the starting phase takes on average .6 ms. When the tumour is however made up of tens of thousands of cancer cells, such as shown in figure 4.4, the irradiation takes on average 1.7 ms.

4.2 Training of the agents

Before introducing the results of the agents' training and their proposed treatments, a baseline treatment will be introduced in order to determine the advantage of these optimized treatments compared to conventional radiation therapy.

Then, for each agent, details will first be given about its training such as its hyperparameters, the number of epochs during which it has been trained and its evolution during training. These hyperparameters have been chosen by conducting a grid search to select the initial learning rate and how it should be adapted during the training, and then by selecting other parameters such as the exploration strategy and the observation type according to which combination reached the best results.

Unlike supervised learning in which, during training, the measured loss should tend to converge towards a minimum if the hyperparameters were correctly tuned, the measured loss of neural networks using reinforcement learning will not necessarily decrease in a stable manner, as the agent can reach states that it has yet to explore after many steps of training, which will make it unable to accurately predict the value of a possible action.

Furthermore, the goal of the agent is to find policies that lead to higher scores, rather than simply being able to accurately estimate action values. Although these two goals are closely connected, an agent that achieves a higher average score on episodes should thus be favoured to an agent that achieves lower losses on training batches. For this reason, the metric that is observed during training will be the average score obtained on a test epoch. Such test epochs will be run to estimate the agent's current score after every training epoch. The neural network that will

¹Higher doses will eliminate more cells and therefore more objects will need to be deleted but the measured time difference was insignificant

be saved and evaluated will be the one that lead to the highest average score during a test epoch.

To be able to compare the agents' treatments to the baseline and other agents despite them using different reward functions, it is necessary to introduce performance indicators that can be computed for all proposed treatments.

These indicators are:

- The TCP
- The average radiation dose used in successful treatments
- The average number of fractions used in successful treatments
- The average duration of successful treatments
- The number of healthy cells that were killed by radiation during successful treatments

This last indicator aims to quantify the damage done during the treatment to healthy tissue surrounding the tumour.

To measure these indicators, averages will be taken over 500 separate treatments.

Finally, for each agent and the baseline, an example treatment will be exposed by first showing the doses and resting times that are chosen during the treatment and how the number of cancer cells evolves. This number of cancer cells is shown in log scale as in the second half of each treatment, the agent is attempting to kill less than 100 remaining cancer cells. As this number sits at around 5000 before the treatment starts, the number of cancer cells would be too small on the graph to see the effects of the doses in the second half of the treatment. Using a logarithmic scale solves this problem

Then, images will show the the tumour's evolution and how the dose is deposited on the grid at different stages of the treatment. The first stage will correspond to the start of the treatment and the last stage to the end of the treatment when the tumour has been eliminated and the dose map is complete. The images of the tumour are obtained using the "densities" observation type. Healthy cells are shown in green and cancer cells are shown in red, with lighter colors indicating higher densities. Black pixels indicate that there are no cells on the corresponding pixel on the grid.

4.2.1 Baseline

The baseline treatment to which the agents will be compared is the one that was used to tune parameters of the simulation. A daily dose of 2 Gy is delivered to the agent in 35 fractions. Afterwards, if there are still cancer cells on the grid, they will again form a tumour and the treatment will therefore have failed.

Evaluation

- TCP: 72.2 %
- Total dose: 60.74 Gy
- Fractions: 30.37
- Duration: 728.91 hours
- Healthy cells killed by radiation: 33166.93

Example of treatment

Figure 4.5 shows the evolution of the number of cancer cells during the treatment. Daily doses of 2 Gy make this number decrease slowly throughout the treatment.

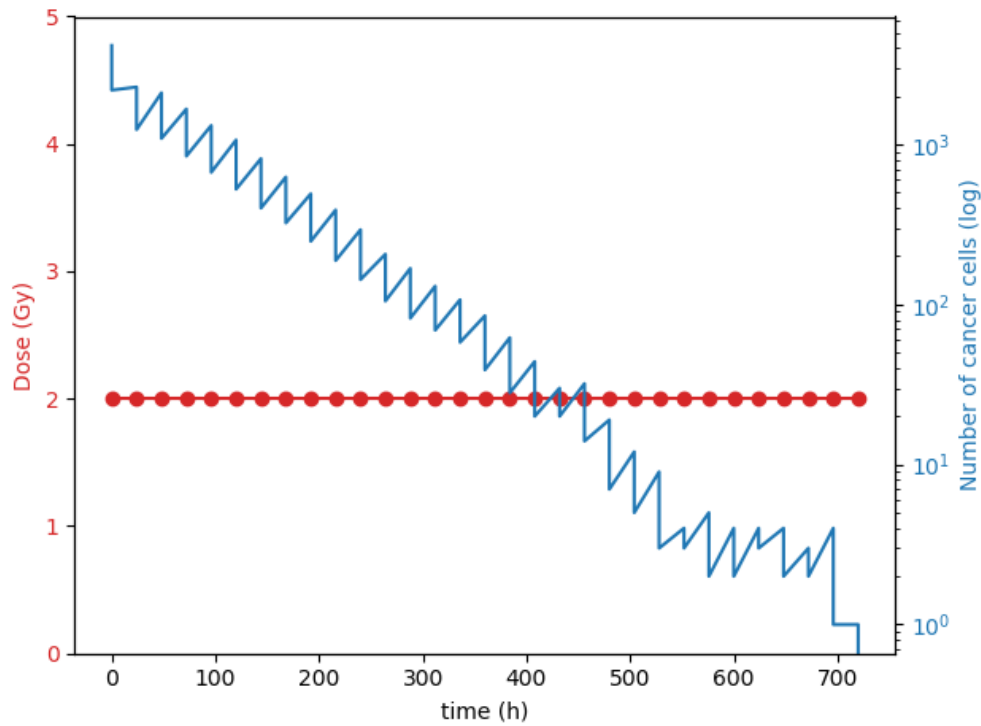


Figure 4.5: Baseline treatment and its effect on the number of cancer cells

Figure 4.6 shows how this treatment affects the tumour and how the dose is deposited on the grid. At the start of the treatment, the relatively low doses don't really reduce the size of the tumour. This means that the radiation will still be deposited in a large radius. The dose map at the end of treatment indeed shows that a big portion of the grid was irradiated with doses of more than 40 Gy.

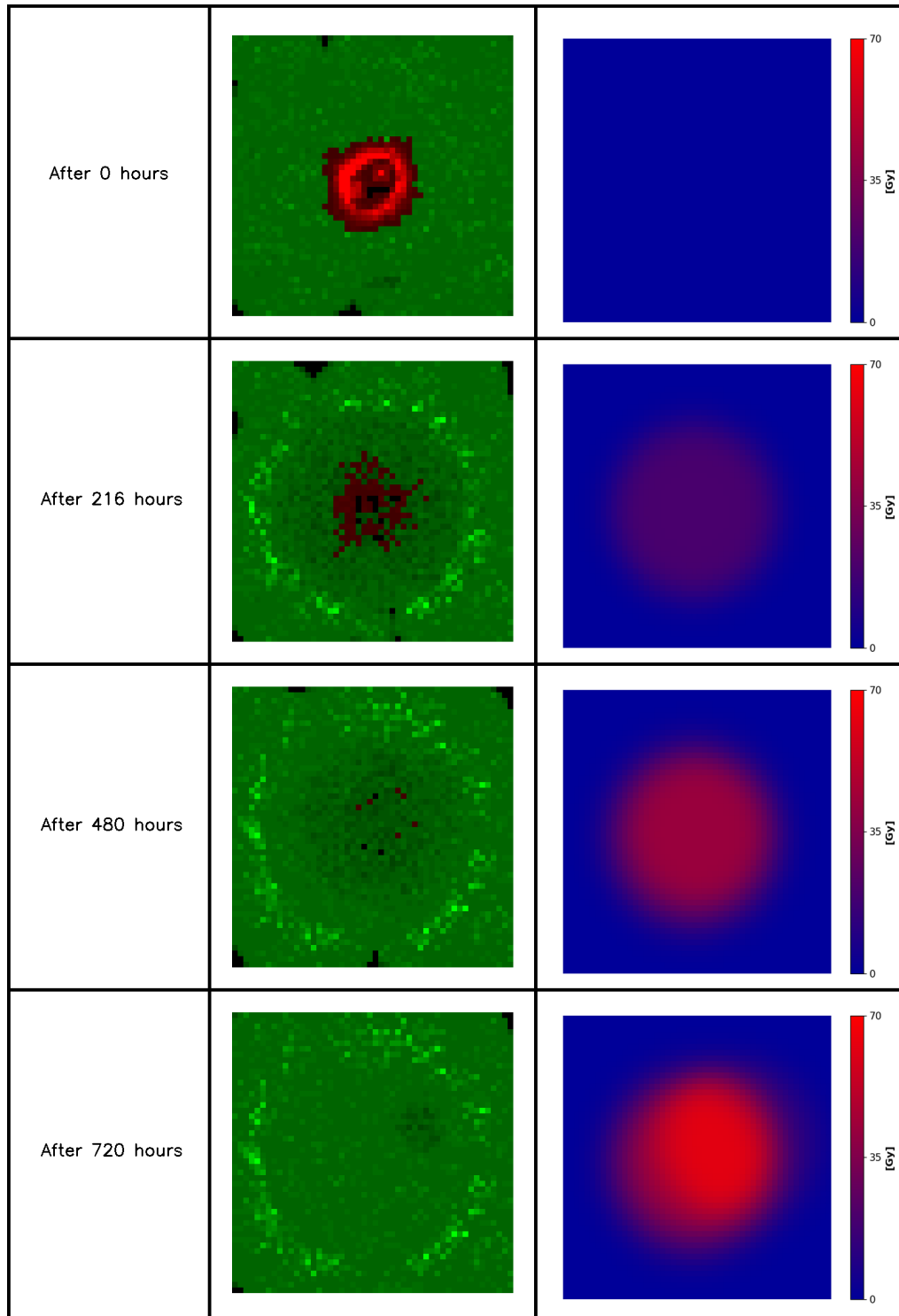


Figure 4.6: Effect of the baseline treatment on a tumour

4.2.2 DQN - killed

This agent was trained using the DQN algorithm and the reward function based on the difference between cancer cells and healthy cells killed. It received observations containing information about the densities of cells. It had to choose actions in the form of doses in a discrete domain going from 0 to 4 Gy with steps of .5 Gy. Between each dose, there was a rest of 24 hours.

Training

The agent was trained for 200 epochs of 2500 timesteps. It used an initial learning rate of 0.01 that was multiplied by 0.9 at the end of each epoch. It used the ϵ -greedy algorithm as an exploration strategy, with ϵ being decreased linearly from 0.8 at the start of the training to 0.2 at the end. The discount factor was set to 0.95 throughout the training.

Figure 4.7 shows the evolution of the average score obtained by the agent on test epochs during the training. It reached a maximum of 0.150 after 37 epochs or 92 500 training steps. The neural network was therefore saved after this test epoch, and the treatment that achieved this average score is evaluated in the next section.

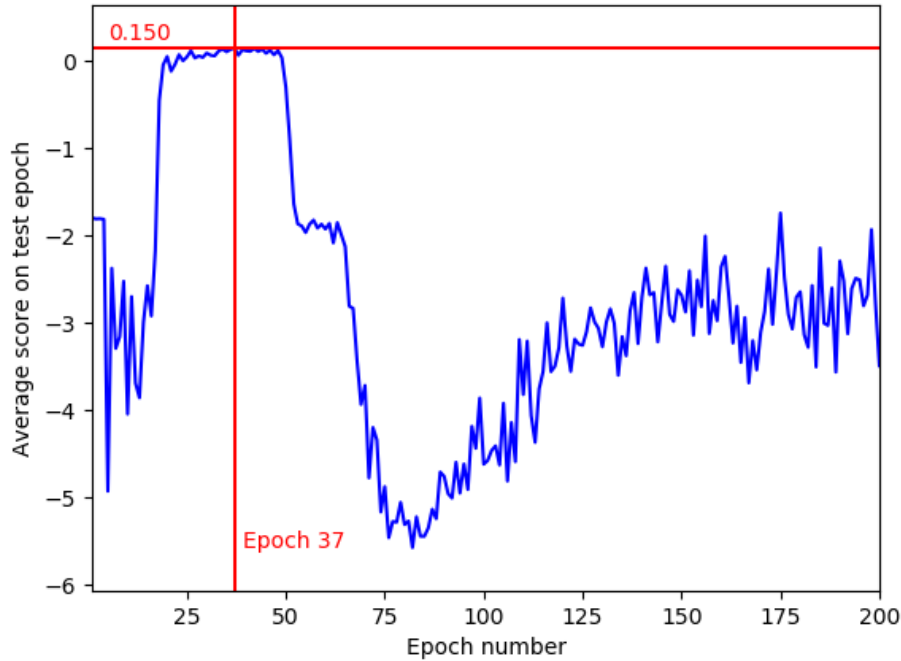


Figure 4.7: Evolution of the score of the DQN - killed agent during its training

Evaluation

- TCP: 100 %

- Total dose: 34.70 Gy
- Fractions: 10.22
- Duration: 245.86 hours
- Healthy cells killed by radiation: 7870.92

Example of treatment

Figure 4.8 shows the treatment proposed by the agent and how it impacts the number of cancer cells in the simulation. It will first send a few high doses of 4 Gy that allow to reduce this number to less than 10. Once that is done, it will eliminate the remaining cancer cells with lower doses of 2.5 Gy.

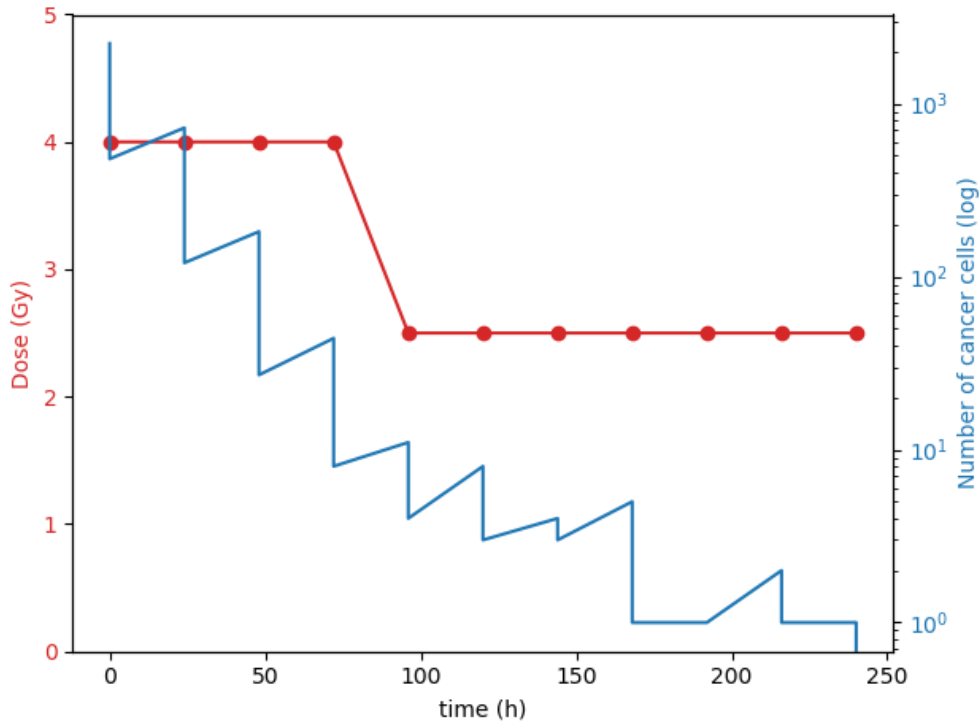


Figure 4.8: Treatment proposed by the DQN - killed agent and its effect on the number of cancer cells

Figure 4.9 shows the effect of this treatment on a tumour and how the dose is deposited on the grid. The 4 high doses at the start of the treatment almost eliminate the tumour completely, and the remaining cancer cells are all in a small radius close to the center of the grid. Therefore, while the agent eliminates the remaining cancer cells with smaller doses, the healthy tissue slowly recovers the space left due to the absence of the tumour. On the dose map, two different circles can be noticed. The larger one corresponds to the initial high doses, while the small one corresponds to the lower doses that were sent in a smaller radius and that allowed to spare most of the grid from the radiation.

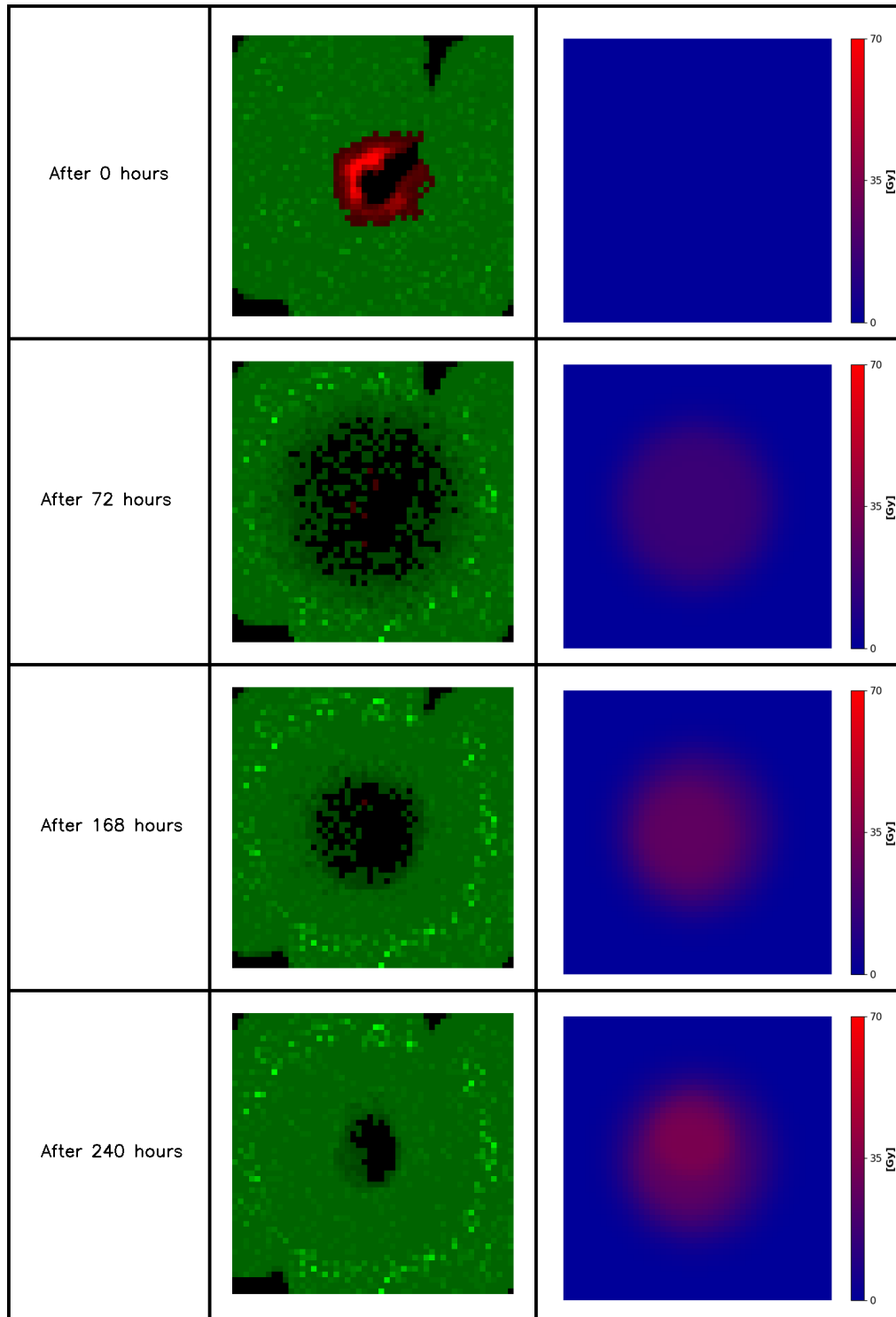


Figure 4.9: Effect of the DQN - killed treatment on a tumour

4.2.3 DDPG - killed

This agent was trained using the DDPG algorithm and the reward function based on the difference between cancer cells and healthy cells killed. It received observations containing information about the densities of cells. It had to choose actions in the form of doses in a continuous domain going from 1 to 5 Gy and resting times between doses in a continuous domain going from 12 to 72 hours.

Training

The agent was trained for 100 epochs of 2500 timesteps. It used an initial learning rate of 0.005 that was multiplied by 0.9 at the end of each epoch. It used the ϵ -greedy algorithm as an exploration strategy, with ϵ being decreased linearly from 0.8 at the start of the training to 0.2 at the end. The discount factor was set to 0.95 throughout the training.

Figure 4.10 shows the evolution of the average score obtained by the agent on test epochs during the training. It reached a maximum of 0.411 after 13 epochs or 32 500 training steps. The neural network was therefore saved after this test epoch, and the treatment that achieved this average score is evaluated in the next sections.

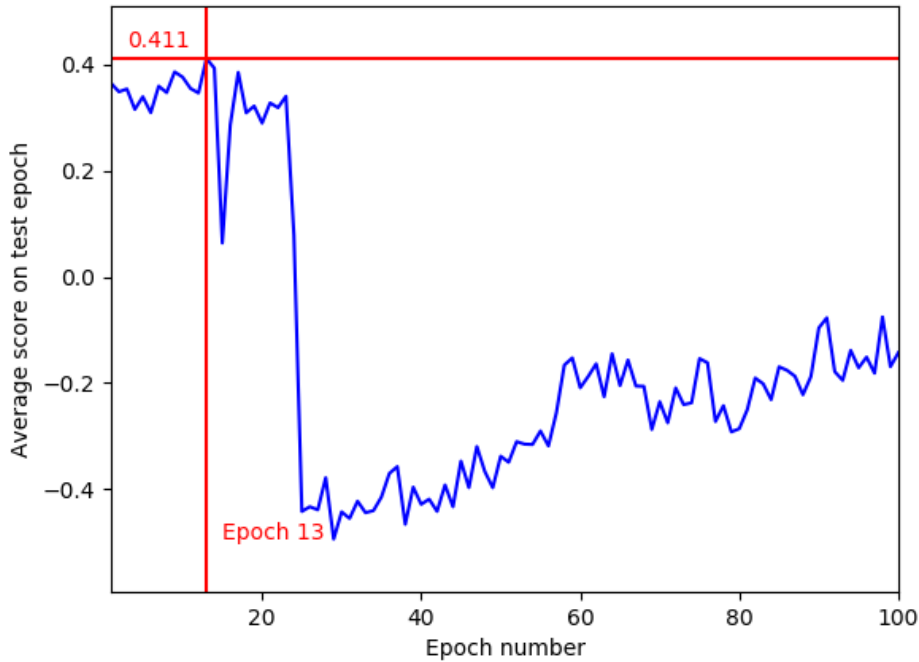


Figure 4.10: Evolution of the score of the DDPG - killed agent during its training

Evaluation

- TCP: 100 %
- Total dose: 38.35 Gy

- Fractions: 34.94
- Duration: 419.48 hours
- Healthy cells killed by radiation: 9768.85

Example of treatment

Figure 4.11 shows the treatment proposed by the agent and how it impacts the number of cancer cells in the simulation. The agent chooses to send a dose a little higher than 1 Gy every 12 hours. This corresponds to a similar daily dose as the baseline, but doesn't give enough time for the tumour to repair from any radiation, which means that the "rebound" effect of the number of cancer cells that can be seen in the figure 4.5 for the baseline is less intense here.

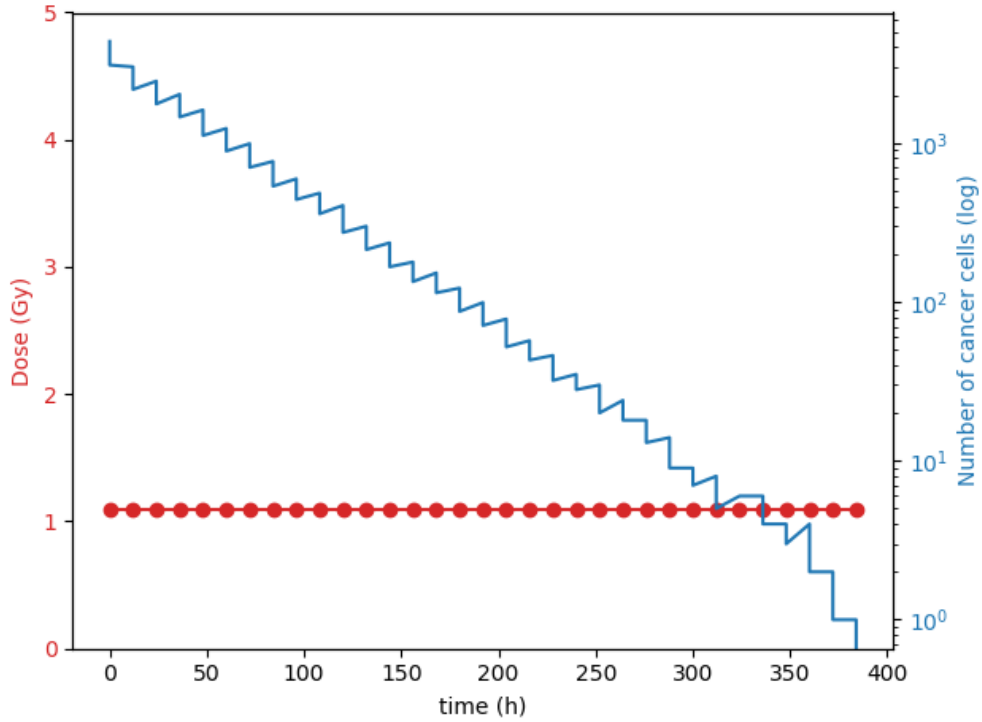


Figure 4.11: Treatment proposed by the DDPG - killed agent and its effect on the number of cancer cells

Figure 4.12 shows the effect of this treatment on a tumour and how the dose is deposited on the grid. After 120 hours of the treatment corresponding to 10 doses, the tumour keeps its initial shape, but we can see that there are no more zones with high densities of cancer cells. At the end of the treatment, we can see that the zone that contained the core of the initial tumour was strongly irradiated and has not yet been reclaimed by healthy tissue.

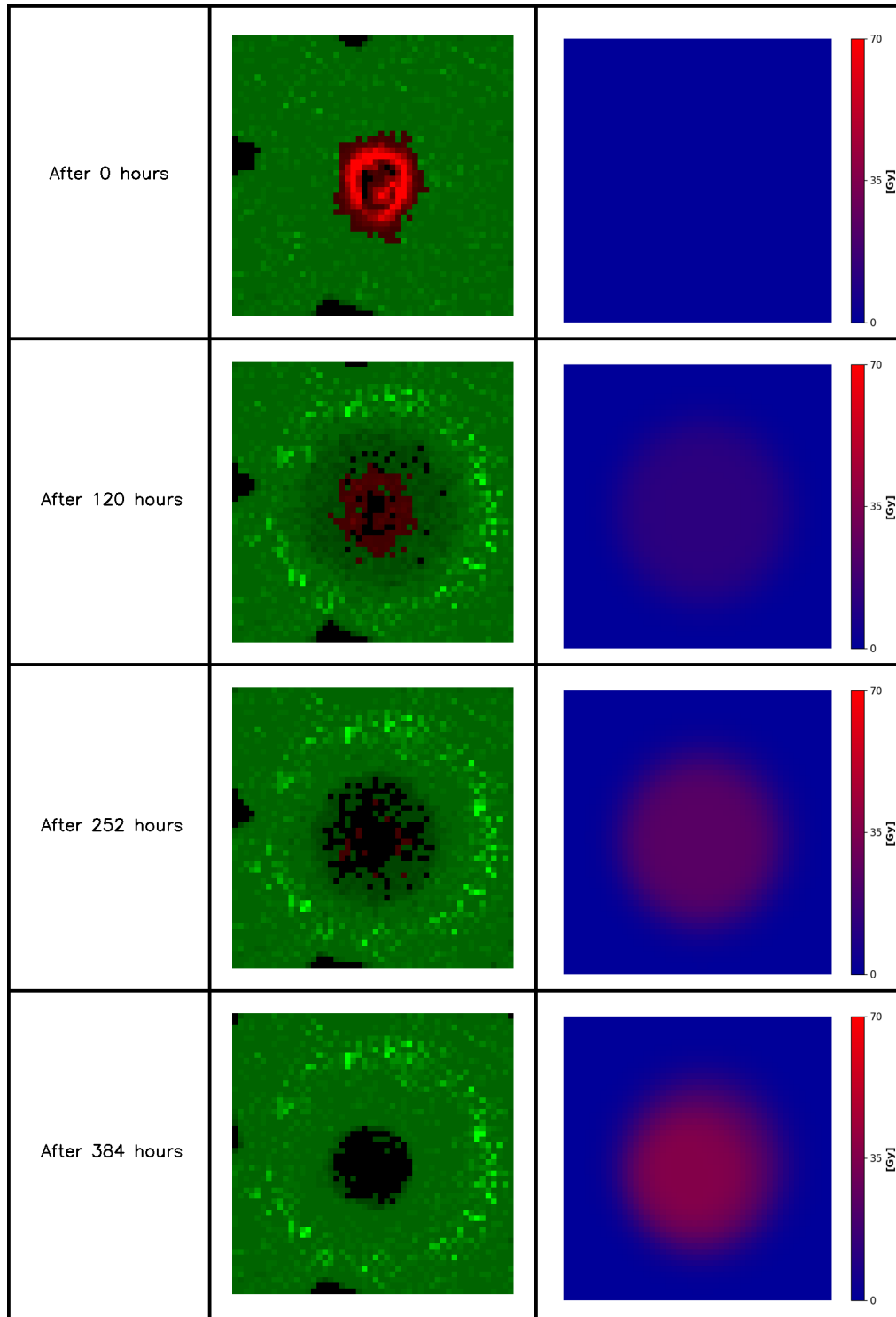


Figure 4.12: Effect of the DDPG - killed treatment on a tumour

4.2.4 DQN - dose

This agent was trained using the DQN algorithm and the reward function based on minimizing the radiation dose. It received observations containing information about the densities of cells. It had to choose actions in the form of doses in a discrete domain going from 0 to 4 Gy with steps of .5 Gy. Between each dose, there was a rest of 24 hours.

Training

The agent was trained for 200 epochs of 2500 timesteps. It used an initial learning rate of 0.01 that was multiplied by 0.9 at the end of each epoch. It used the ϵ -greedy algorithm as an exploration strategy, with ϵ being decreased linearly from 0.8 at the start of the training to 0.2 at the end. The discount factor was set to 0.95 throughout the training.

Figure 4.13 shows the evolution of the average score obtained by the agent on test epochs during the training. It reached a maximum of -0.389 after 42 epochs or 105 000 training steps. The neural network was therefore saved after this test epoch, and the treatment that achieved this average score is evaluated in the next sections. The graph of the score shows that the training was quite unstable.

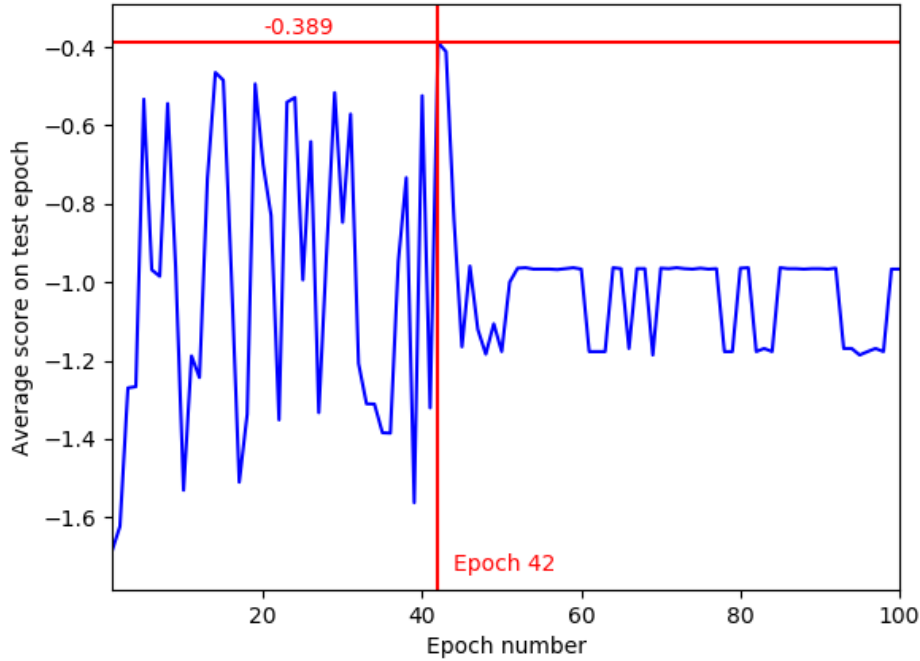


Figure 4.13: Evolution of the score of the DQN - dose agent during its training

Evaluation

- TCP: 100 %

- Total dose: 39.19 Gy
- Fractions: 13.15
- Duration: 317.86 hours
- Healthy cells killed by radiation: 11951.92

Example of treatment

Figure 4.14 shows the treatment proposed by the agent and how it impacts the number of cancer cells in the simulation. The agent chooses to send a dose of 3 Gy every 24 hours.

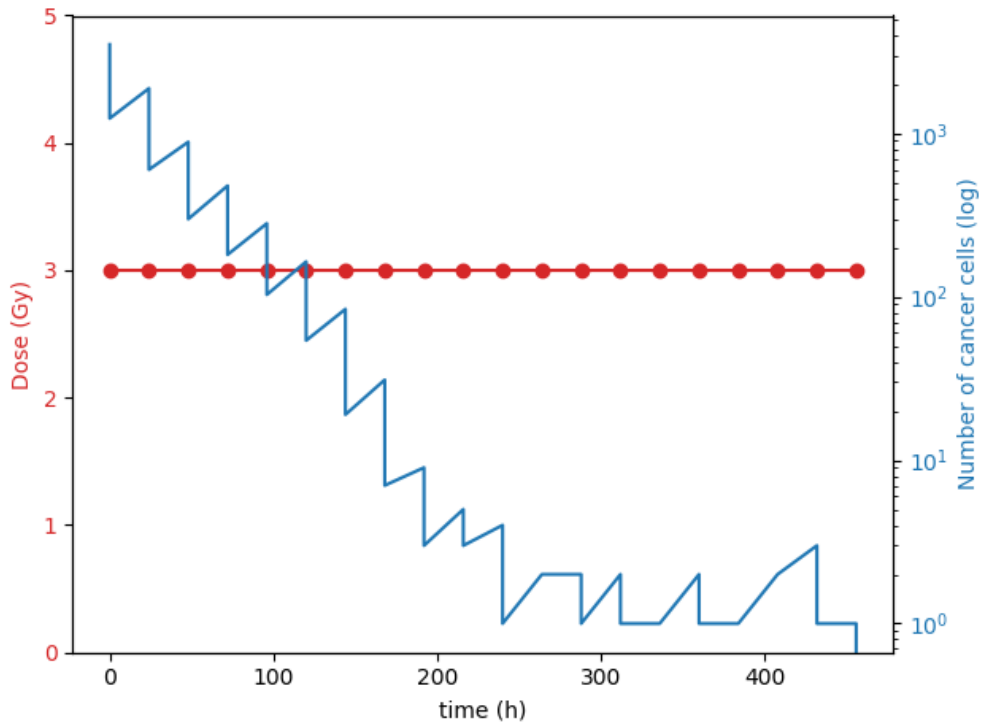


Figure 4.14: Treatment proposed by the DQN - dose agent and its effect on the number of cancer cells

Figure 4.15 shows the effect of this treatment on a tumour and how the dose is deposited on the grid. We can see that after 144 hours or 7 doses of 3 Gy, the initial tumour is almost completely destroyed. The remaining cells are all in a smaller radius and are destroyed over the rest of the treatment. The final dose map shows these two different radii, with the smaller one having received a dose of around 40 Gy.

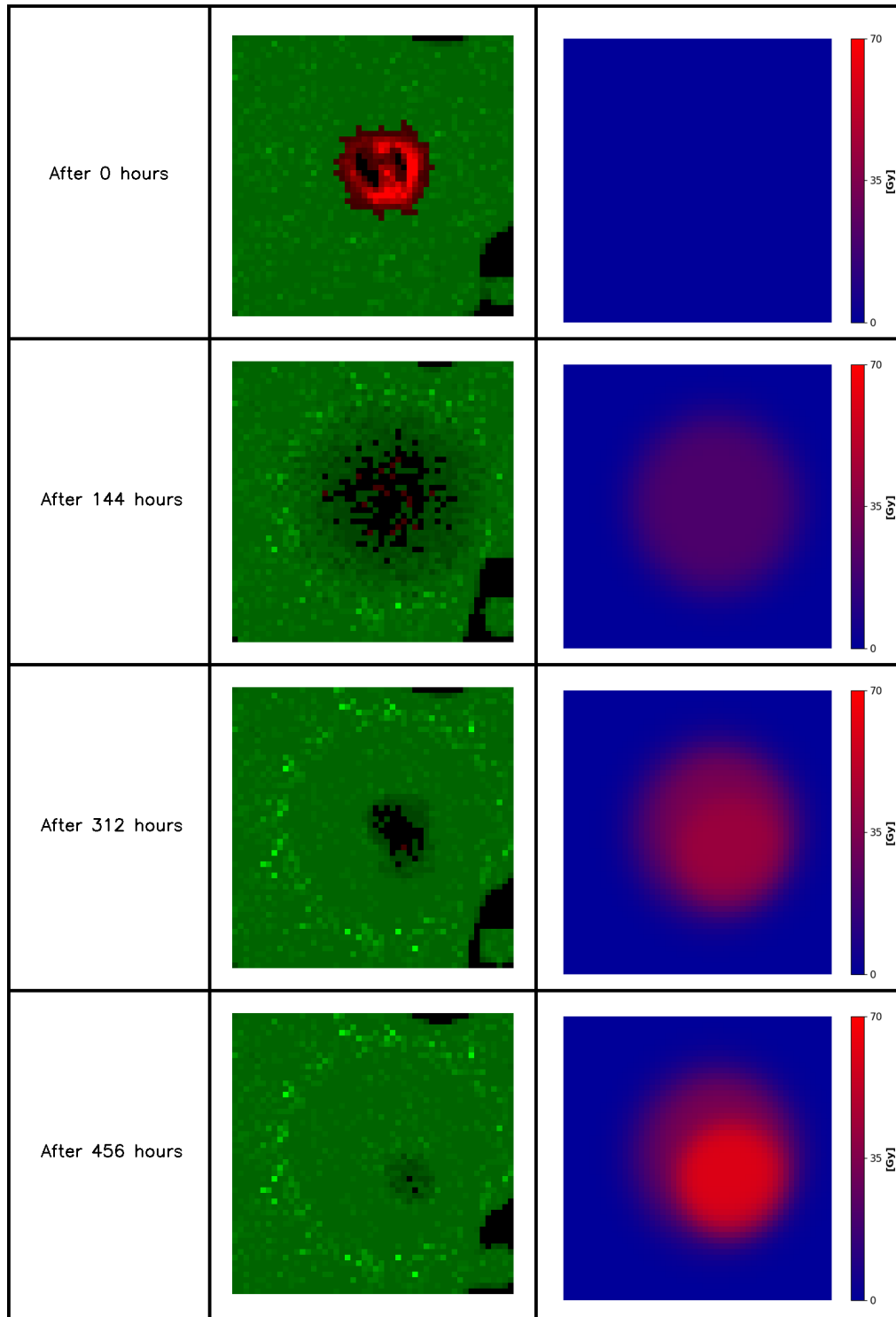


Figure 4.15: Effect of the DQN - dose treatment on a tumour

4.2.5 DDPG - dose

This agent was trained using the DDPG algorithm and the reward function based on minimizing the dose. It received observations containing information about the densities of cells. It had to choose actions in the form of doses in a continuous domain going from 1 to 5 Gy and resting times between doses in a continuous domain going from 12 to 72 hours.

Training

The agent was trained for 200 epochs of 2500 timesteps. It used an initial learning rate of 0.01 that was multiplied by 0.9 at the end of each epoch. It used the ϵ -greedy algorithm as an exploration strategy, with ϵ being decreased linearly from 0.8 at the start of the training to 0.2 at the end. The discount factor was set to 0.95 throughout the training.

Figure 4.16 shows the evolution of the average score obtained by the agent on test epochs during the training. It reached a maximum of -0.251 after 135 epochs or 337 500 training steps. The neural network was therefore saved after this test epoch, and the treatment that achieved this average score is evaluated in the next sections.

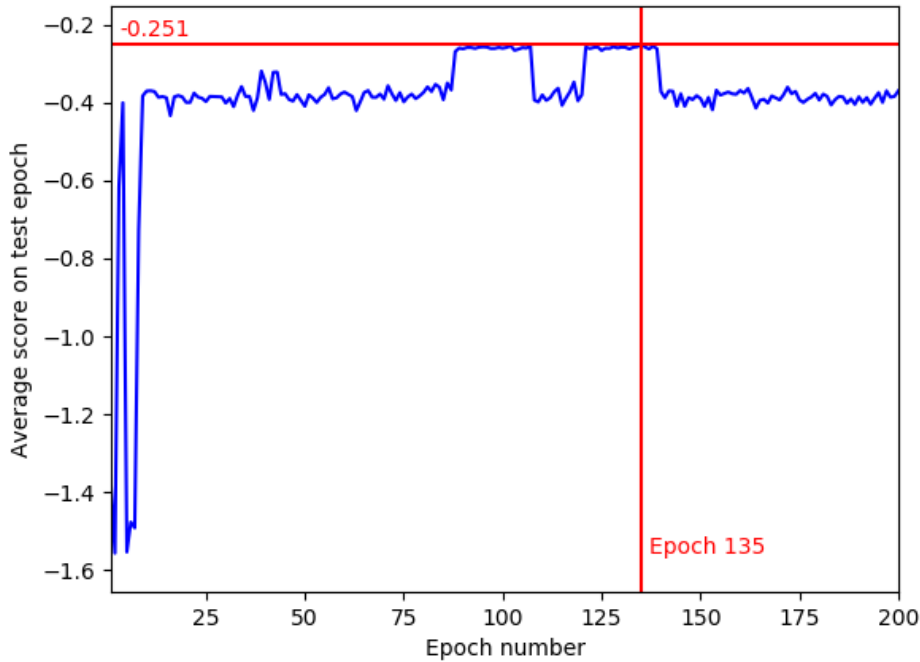


Figure 4.16: Evolution of the score of the DDPG - dose agent during its training

Evaluation

- TCP: 100 %
- Total dose: 26.08 Gy

- Fractions: 5.30
- Duration: 63.78 hours
- Healthy cells killed by radiation: 4391.43

Example of treatment

Figure 4.17 shows the treatment proposed by the agent and how it impacts the number of cancer cells in the simulation. The agent chooses to send a dose a little lower than 5 Gy every 12 hours. The number of cancer cells decreases very fast and reaches 0 in only 60 hours or two and a half days.

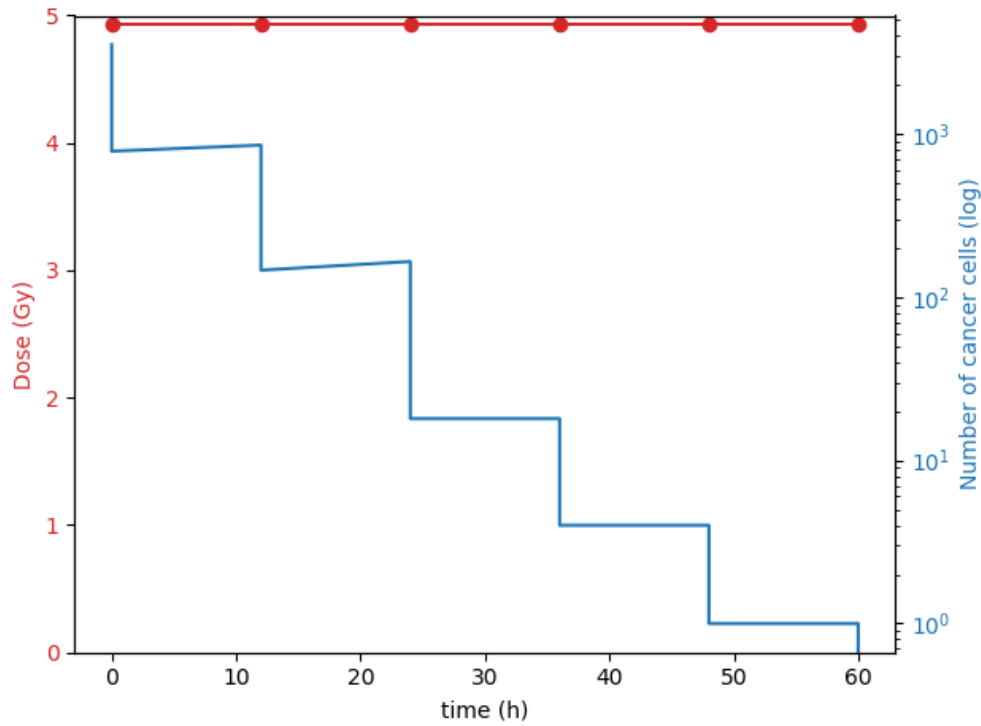


Figure 4.17: Treatment proposed by the DDPG - dose agent and its effect on the number of cancer cells

Figure 4.18 shows the effect of this treatment on a tumour and how the dose is deposited on the grid. After only 36 hours, the tumour has almost completely disappeared, and a large portion of the grid is now empty. At the end of the treatment, healthy cells have started to reclaim this empty space. The dose map shows that there are no zones on the grid that received a particularly high dose of radiation.

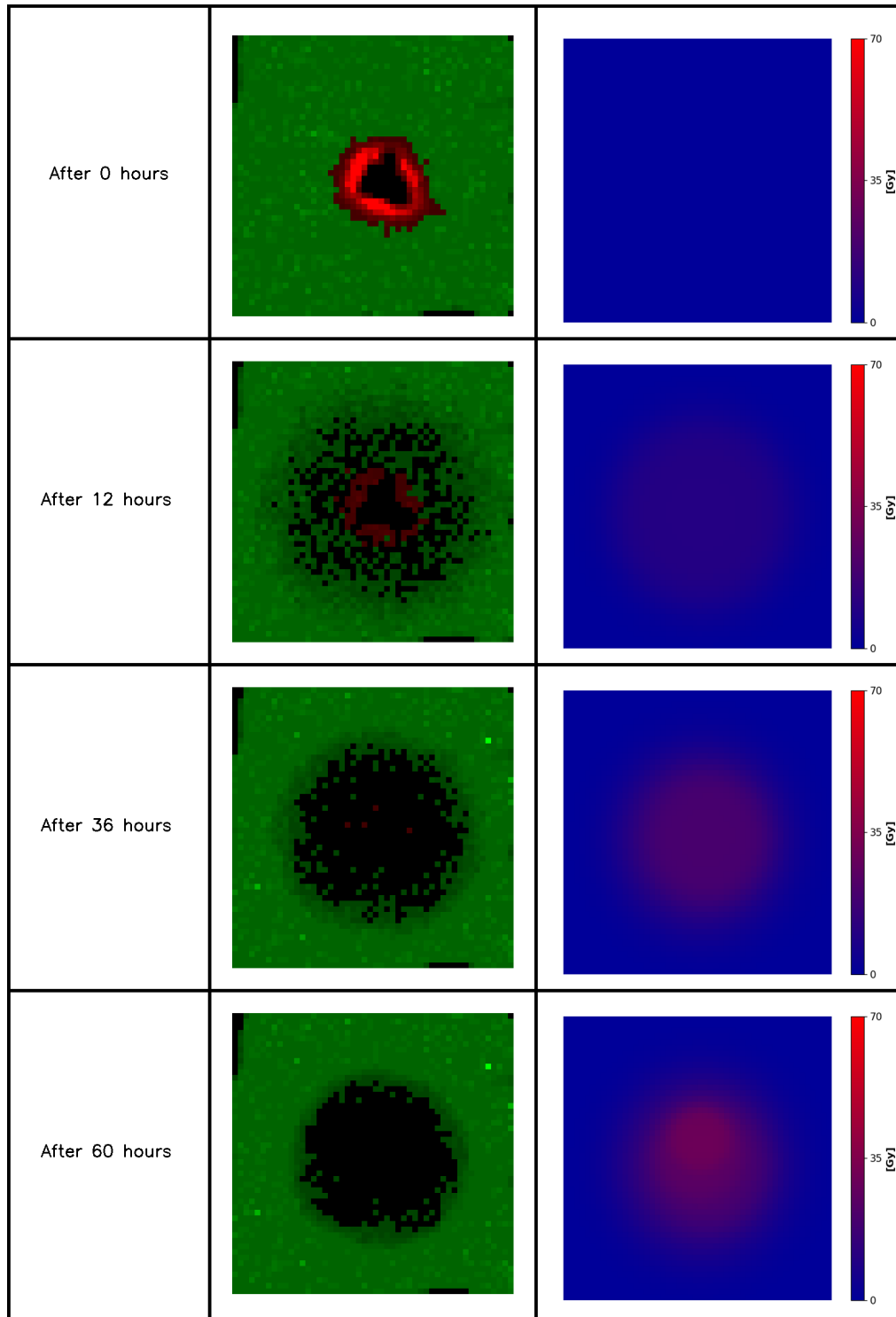


Figure 4.18: Effect of the DDPG - dose treatment on a tumour

4.3 Discussion of the results

	TCP (%)	Dose (Gy)	Duration (h)	Fractions	HCells killed
Baseline	72.2	60.74	728.91	30.37	33166.93
DQN - killed	100	34.70	245.86	10.22	7870.92
DDPG - killed	100	38.35	419.48	34.94	9768.85
DQN - dose	100	39.19	317.86	13.15	11951.92
DDPG - dose	100	26.08	63.78	5.30	4391.43

Table 4.1: Summary of the results

Table 4.1 summarizes the evaluation indicators obtained by the baseline treatment and the treatments found by the agents to optimize their reward function. The agents found treatments that provide much better results than the baseline on the simulation. They all reach a TCP of 100 %, administer a smaller total dose, with a treatment shorter in duration and with less fractions. All of their proposed treatments also killed significantly less healthy cells through radiation.

The agent that used the dose reward function and the DDPG algorithm obtained the best result for every indicator.

Both agents using the dose reward function choose to send high doses(almost the highest for DDPG) at every timestep. They manage to avoid doing too much damage to healthy tissue with those high doses because the dose deposition function, described in section 3.2.5, is very precise and can thus target the current zone occupied by the tumour accurately while avoiding the irradiation of healthy cells that don't directly border this tumour. This choice of high doses thus seems to indicate that treatments with higher dose fractions could provide better outcomes assuming that an accurate form of radiation dose deposition is used, as is the case in proton therapy.

Both agents that were trained using the DDPG algorithm suggest treatments requiring a dose to be delivered every 12 hours. While that is physically possible, it is quite inconvenient for the patients being treated. Indeed, during conventional treatments, patients have to come only once a day to a radiation therapy center. If the time between doses is reduced, patients might be required to stay at this center throughout the treatment, which could be logistically impossible.

The two agents that used the DQN algorithm send a dose every 24 hours as is the case in conventional RT. It is interesting to note that the DQN agent that attempted to minimize the dose achieved a higher average dose than the agent using the reward function based on cancer cells and healthy cells killed. This indicates that the agent has not fully converged and that it should be trained more as there are policies that should lead it to higher scores on its reward function.

One of the factors that might have lead this agent to not fully converge is that its training was not very stable. This likely happened because the reward function that it uses requires it to estimate values in the long term. Indeed, as the agent's

rewards are proportional to - dose, the agent will want to send a minimum dose of 0 Gy to maximize its immediate reward. To avoid this, a special reward of -1 was introduced at the end of episodes corresponding to failed treatments. However, at the start of an episode, the agent is at least 25 timesteps away from this special reward, and it is thus complex for it to have a big impact on the estimated values for the agent, since it will be multiplied at least 25 times by the discount factor.

To solve this problem, the discount factor could be slightly increased to give more importance to long term rewards, or a special reward could also be returned at the end of successful treatments as the agent will see a higher value in sending radiation doses throughout an episode. This however means that the agent's goal will no longer only be to minimize the dose.

We can also see that in certain treatments, either the maximum or the minimum possible dose (or close to it in the case of DDPG) is chosen to be administered. It would be interesting to conduct experiments where the action space is slightly increased in the direction of the minimum or maximum dose, in order to determine which dose would be chosen by the agents if the intervals in which the agent have to choose didn't have limits, as this leaves them more freedom to optimize their reward function.

Finally, it is important to note that the number of cancer cells killed by radiation during a treatment is not a good indicator to represent the NTCP. Indeed, we can see that agents that allow healthy cells to proliferate in the space left after the destruction of the tumour are penalized because these new healthy cells are killed by radiation during the treatment. Meanwhile, the treatment suggested by the DDPG - dose agent creates a zone regularly irradiated by high doses and in which healthy cells can thus not manage to proliferate. This decreases the number of healthy cells that can possibly be killed by radiation during this treatment.

This situation does not correspond to real life as cells that have been fatally damaged by radiation simply disappear from the grid in the simulation, while real cells they would enter a state of necrosis in which they will slowly disintegrate which will not allow healthy tissue to directly proliferate in the space of the damaged cell.

Because of this inaccuracy as well as other effects missing from the model as explained in section 3.2.6, the simulation doesn't accurately predict the damage that the treatments suggested by the agents would do to healthy tissues surrounding the tumour.

Chapter 5

Conclusion and future work

The objective of this master’s thesis was to explore the possibility of training deep reinforcement learning agents on a mathematical model of cancer development in healthy tissue in order to optimize dose planification for radiation therapy. A simulation was developed and agents were trained with two different deep RL algorithms and with two different reward functions. The treatments found by the agents were compared to a baseline corresponding to what is conventionally used in radiation therapy today. They were found to achieve significantly better results, with their treatments reaching 100 % success in eliminating the tumour and using smaller doses of radiation in shorter treatments.

However, the side effects of these treatments on healthy tissue are likely to be underestimated as the simulation was implemented to be efficient and some important effects, such as the bystander effect, are omitted. Furthermore, organs and normal tissues that normally surround a tumour are represented in the simulation as an almost uniform layer of healthy cells. This doesn’t reflect the situation of RT where organs at risk of side effects are clearly identified on scans before the start of the treatment, with complication probabilities being computed. In the simulation, there is no way to correctly estimate the effect that the current treatment would have on real tissue and it is not possible to predict the NTCP.

Nevertheless, the results are promising and show that this approach is worth pursuing. In particular, the agent that uses a reward function based on maximizing the number of cancer cells killed while minimizing the number of healthy cells killed optimized its reward function by selecting high doses at the start of the treatment to stop the growth of the tumour and eliminate a large portion of it in a short period, then selecting lower doses during the rest of the treatment to eliminate the remnants of this tumour while surrounding normal tissue is allowed to repair. It would be interesting to see if this treatment could also be found using more complex simulations and other reward functions.

In order to build on this project and find more significant results in the future, the first step to take would be to use a more complex and realistic simulation. Ideally, such a simulation would represent a tumour and surrounding organs in three

dimensions, and model the effect of radiation on these elements more accurately. Angiogenesis, hypoxia, bystander effects and different phenotypes of cancer cells should also be included in this simulation. This could however make it too slow to be used in conjunction with a deep RL agent. To solve this problem, one approach that could be used would be to implement two similar simulations: a simple one, that only includes the most important effects and that can be used for an agent's training, and a more complex one, including all the phenomena previously mentioned, that could be used to evaluate an agent's proposed treatment.

This simulation should be built or validated using clinical data that show real examples of how tumours develop and invade healthy tissues, and how they react to radiation.

Finally, even with more accurate simulations, it would be quite important to first conduct clinical experiments to validate the treatments proposed by the trained agents. Indeed, many of these treatments are unconventional, and it is possible that they would result in unforeseen side effects as all simulations rely on assumptions and can not perfectly reflect the complexity of human biology.

Bibliography

- [1] *Cancer*, en. [Online]. Available: <https://www.who.int/news-room/fact-sheets/detail/cancer> (visited on 06/06/2020).
- [2] *Radiation Therapy for Cancer - National Cancer Institute*, en, cgvArticle, Apr. 2015. [Online]. Available: <https://www.cancer.gov/about-cancer/treatment/types/radiation-therapy> (visited on 06/06/2020).
- [3] P. Paul-Gilloteaux, V. Potiron, G. Delpon, S. Supiot, S. Chiavassa, F. Paris, and S. V. Costes, “Optimizing radiotherapy protocols using computer automata to model tumour cell death as a function of oxygen diffusion processes,” en, *Scientific Reports*, vol. 7, no. 1, pp. 1–14, May 2017, ISSN: 2045-2322. DOI: 10.1038/s41598-017-01757-6. [Online]. Available: <https://www.nature.com/articles/s41598-017-01757-6>.
- [4] I. Shuryak, E. J. Hall, and D. J. Brenner, “Optimized Hypofractionation Can Markedly Improve Tumor Control and Decrease Late Effects for Head and Neck Cancer,” en, *International Journal of Radiation Oncology*Biophysics*, vol. 104, no. 2, pp. 272–278, Jun. 2019, ISSN: 0360-3016. DOI: 10.1016/j.ijrobp.2019.02.025. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0360301619302512>.
- [5] A. Jalalimanesh, H. Shahabi Haghighi, A. Ahmadi, and M. Soltani, “Simulation-based optimization of radiotherapy: Agent-based modeling and reinforcement learning,” *Mathematics and Computers in Simulation*, Jun. 2016. DOI: 10.1016/j.matcom.2016.05.008.
- [6] Grégoire Moreau, *Radio_rl*, original-date: 2019-06-18T07:46:13Z, Jun. 2020. [Online]. Available: https://github.com/gregoire-moreau/radio_rl (visited on 06/07/2020).
- [7] F. Michor and K. Beal, “Improving Cancer Treatment via Mathematical Modeling: Surmounting the Challenges is Worth the Effort,” *Cell*, vol. 163, no. 5, pp. 1059–1063, Nov. 2015, ISSN: 0092-8674. DOI: 10.1016/j.cell.2015.11.002. [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4676401/>.
- [8] J. Metzcar, Y. Wang, R. Heiland, and P. Macklin, “A Review of Cell-Based Computational Modeling in Cancer Biology,” *JCO Clinical Cancer Informatics*, no. 3, pp. 1–13, Feb. 2019. DOI: 10.1200/CCI.18.00069. [Online]. Available: <https://ascopubs.org/doi/full/10.1200/CCI.18.00069>.
- [9] A. R. A. Anderson, K. A. Rejniak, P. Gerlee, and V. Quaranta, “Microenvironment driven invasion: A multiscale multimodel investigation,” en, *Journal of Mathematical Biology*, vol. 58, no. 4, p. 579, Oct. 2008, ISSN: 1432-1416.

- DOI: 10.1007/s00285-008-0210-2. [Online]. Available: <https://doi.org/10.1007/s00285-008-0210-2>.
- [10] K. Smallbone, R. A. Gatenby, R. J. Gillies, P. K. Maini, and D. J. Gavaghan, "Metabolic changes during carcinogenesis: Potential impact on invasiveness," en, *Journal of Theoretical Biology*, vol. 244, no. 4, pp. 703–713, Feb. 2007, ISSN: 0022-5193. DOI: 10.1016/j.jtbi.2006.09.010. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0022519306004115>.
- [11] S. R. McDougall, A. R. A. Anderson, and M. A. J. Chaplain, "Mathematical modelling of dynamic adaptive tumour-induced angiogenesis: Clinical implications and therapeutic targeting strategies," en, *Journal of Theoretical Biology*, vol. 241, no. 3, pp. 564–589, Aug. 2006, ISSN: 0022-5193. DOI: 10.1016/j.jtbi.2005.12.022. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0022519305005564>.
- [12] N. O’Neil, "An agent based model of tumor growth and response to radiotherapy," Master’s thesis, Virginia Commonwealth University, Richmond, Virginia, May 2012.
- [13] C. M. van Leeuwen, A. L. Oei, J. Crezee, A. Bel, N. a. P. Franken, L. J. A. Stalpers, and H. P. Kok, "The alfa and beta of tumours: A review of parameters of the linear-quadratic model, derived from clinical radiotherapy studies," eng, *Radiation Oncology (London, England)*, vol. 13, no. 1, p. 96, May 2018, ISSN: 1748-717X. DOI: 10.1186/s13014-018-1040-z.
- [14] G. G. Powathil, D. J. A. Adamson, and M. A. J. Chaplain, "Towards Predicting the Response of a Solid Tumour to Chemotherapy and Radiotherapy Treatments: Clinical Insights from a Computational Model," *PLoS Computational Biology*, vol. 9, no. 7, Jul. 2013, ISSN: 1553-734X. DOI: 10.1371/journal.pcbi.1003120. [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3708873/>.
- [15] V. Francois-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An Introduction to Deep Reinforcement Learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018, arXiv: 1811.12560, ISSN: 1935-8237, 1935-8245. DOI: 10.1561/22000000071. [Online]. Available: <http://arxiv.org/abs/1811.12560>.
- [16] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of Real-World Reinforcement Learning," *arXiv:1904.12901 [cs, stat]*, Apr. 2019, arXiv: 1904.12901. [Online]. Available: <http://arxiv.org/abs/1904.12901>.
- [17] *Emergent Tool Use from Multi-Agent Interaction*, en, Sep. 2019. [Online]. Available: <https://openai.com/blog/emergent-tool-use/> (visited on 06/07/2020).
- [18] C. Yu, J. Liu, and S. Nemat, "Reinforcement Learning in Healthcare: A Survey," *arXiv:1908.08796 [cs]*, Apr. 2020, arXiv: 1908.08796. [Online]. Available: <http://arxiv.org/abs/1908.08796>.
- [19] H.-H. Tseng, Y. Luo, S. Cui, J.-T. Chien, R. K. Ten Haken, and I. E. Naqa, "Deep reinforcement learning for automated radiation adaptation in lung cancer," en, *Medical Physics*, vol. 44, no. 12, pp. 6690–6705, Dec. 2017, ISSN: 00942405. DOI: 10.1002/mp.12625. [Online]. Available: <http://doi.wiley.com/10.1002/mp.12625>.

- [20] K. M. Ph.D, *Aerobic Respiration*, en-US, Oct. 2016. [Online]. Available: <https://biologydictionary.net/aerobic-respiration/> (visited on 05/28/2020).
- [21] B. Muz, P. de la Puente, F. Azab, and A. K. Azab, “The role of hypoxia in cancer progression, angiogenesis, metastasis, and resistance to therapy,” *Hypoxia*, vol. 3, pp. 83–92, Dec. 2015, ISSN: 2324-1128. DOI: 10.2147/HP.S93413. [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5045092/>.
- [22] *PET Scan: Definition, Purpose, Procedure, and Results*, en. [Online]. Available: <https://www.healthline.com/health/pet-scan> (visited on 06/08/2020).
- [23] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter, “Blood Vessels and Endothelial Cells,” en, *Molecular Biology of the Cell. 4th edition*, 2002. [Online]. Available: <https://www.ncbi.nlm.nih.gov/books/NBK26848/>.
- [24] G. M. Cooper, “The Eukaryotic Cell Cycle,” en, *The Cell: A Molecular Approach. 2nd edition*, 2000. [Online]. Available: <https://www.ncbi.nlm.nih.gov/books/NBK9876/>.
- [25] *Proton Therapy — What Is Proton Therapy — Provision CARES*, en-US. [Online]. Available: <https://provisionhealthcare.com/about-proton-therapy/advantages-of-proton/> (visited on 06/08/2020).
- [26] A. Marín, M. Martín, O. Liñán, F. Alvarenga, M. López, L. Fernández, D. Büchser, and L. Cerezo, “Bystander effects and radiotherapy,” en, *Reports of Practical Oncology & Radiotherapy*, vol. 20, no. 1, pp. 12–21, Jan. 2015, ISSN: 1507-1367. DOI: 10.1016/j.rpor.2014.08.004. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1507136714001424>.
- [27] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018, ISBN: 0262039249.
- [28] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” en, *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14236. [Online]. Available: <http://www.nature.com/articles/nature14236>.
- [29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv:1509.02971 [cs, stat]*, Jul. 2019, arXiv: 1509.02971. [Online]. Available: <http://arxiv.org/abs/1509.02971>.
- [30] V. François-Lavet *et al.*, *Deer*, <https://deer.readthedocs.io/>, 2016.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl